

Verification Continuum™

FPGA Synthesis Reference Manual

March 2021

SYNOPSYS®

<http://solvnet.synopsys.com>

Synopsys Confidential Information

Copyright Notice and Proprietary Information

© 2021 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

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.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at

<http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

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

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 East Middlefield Road
Mountain View, CA 94043
www.synopsys.com

March 2021

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

Contents

Chapter 1: Product Overview

Overview of the Synthesis Tools	20
Common Features	20
Synplify Pro and Synplify Premier Features	21
Synplify Premier Features	22
Synopsys FPGA Tool Features	23
Graphic User Interface	26
Getting Help	31

Chapter 2: User Interface Overview

The Project View	34
Project Management View	38
The Project Results View	41
Project Status Tab	41
Report Tab	46
Timing Report View	48
Implementation Directory	49
Process View	50
Other Windows and Views	53
Dockable GUI Entities	54
Watch Window	54
Tcl Script and Messages Windows	57
Tcl Script Window	58
Message Viewer	58
Output Windows (Tcl Script and Watch Windows)	62
Text Editor View	62
Context Help Editor Window	65
Demos & Examples	67
Interactive Attribute Examples	68
Search SolvNetPlus	70

Using the Mouse	71
Mouse Operation Terminology	71
Using Mouse Strokes	72
Using the Mouse Buttons	73
Using the Mouse Wheel	75
Toolbars	76
Project Toolbar	77
Analyst Toolbar	78
Physical Analyst Toolbar	80
Text Editor Toolbar	80
FSM Viewer Toolbar	81
Tools Toolbar	83
Design Planner Toolbar	83
Keyboard Shortcuts	84
Buttons and Options	92

Chapter 3: HDL Analyst Tool

HDL Analyst Views and Commands	98
RTL View	98
Technology View	100
Hierarchy Browser	102
FSM Viewer Window	103
Filtered and Unfiltered Schematic Views	105
Accessing HDL Analyst Commands	106
Schematic Objects and Their Display	108
Object Information	108
Sheet Connectors	109
Primitive and Hierarchical Instances	110
Transparent and Opaque Display of Hierarchical Instances	111
Hidden Hierarchical Instances	113
Schematic Display	113
Basic Operations on Schematic Objects	117
Finding Schematic Objects	117
Selecting and Unselecting Schematic Objects	118
Crossprobing Objects	119
Dragging and Dropping Objects	121
Multiple-sheet Schematics	122
Controlling the Amount of Logic on a Sheet	122
Navigating Among Schematic Sheets	122

Multiple Sheets for Transparent Instance Details	124
Exploring Design Hierarchy	125
Pushing and Popping Hierarchical Levels	125
Navigating With a Hierarchy Browser	129
Looking Inside Hierarchical Instances	130
Filtering and Flattening Schematics	133
Commands That Result in Filtered Schematics	133
Combined Filtering Operations	134
Returning to The Unfiltered Schematic	134
Commands That Flatten Schematics	135
Selective Flattening	136
Filtering Compared to Flattening	137
Timing Information and Critical Paths	139
Timing Reports	139
Critical Paths and the Slack Margin Parameter	140
Examining Critical Path Schematics	141

Chapter 4: Constraint Guidelines

Constraint Types	144
Constraint Files	145
Timing Constraints	147
FDC Constraints	150
Methods for Creating Constraints	151
Constraint Translation	153
sdc2fdc Conversion	154
sopc2syn Conversion	158
qsf2sdc Conversion	160
UCF Conversion	162
Constraint Checking	176
Database Object Search	178
Forward Annotation	179
Auto Constraints	179

Chapter 5: Input and Result Files

Input Files	182
HDL Source Files	185

Libraries	187
Open Verification Library (Verilog)	188
The Generic Technology Library	188
ASIC Library Files	189
Output Files	191
Log File	199
Timing Reports	205
Timing Report Header	206
Performance Summary	206
Clock Pre-map Reports	208
Clock Relationships	211
Interface Information	212
A synchronous Clock Report	213
Hierarchical Area Report	215
Constraint Checking Report	218
Gated Clock Conversion Report	225

Chapter 6: RAM and ROM Inference

Guidelines and Support for RAM Inference	228
Automatic RAM Inference	229
Block RAM	229
RAM Attributes	231
Block RAM Inference	233
Block RAM Examples	239
LUTRAM Inference	252
LUTRAM Examples	254
RAM Inference with Control Signals	257
RAM with Control Signals Examples	258
Distributed RAM Inference	264
Distributed RAM Examples	267
Asymmetric RAM Inference	269
Asymmetric RAM Examples	269
Asymmetric RAM Inference as a Black Box Model	296
Byte-Enable RAM Inference	302
Byte-Enable RAM Examples	302

UltraRAM Block Inference	305
UltraRAM Examples	305
Initial Values for RAMs	310
Example 1: RAM Initialization	310
Example 2: Cross-Module Referencing for RAM Initialization	311
Initialization Data File	313
Forward Annotation of Initial Values	316
Initial Values for Asymmetric RAM	317
RAM Instantiation with SYNCORE	326
ROM Inference	327

Chapter 7: SynCore IP Tool

SYNCORE FIFO Compiler	334
Synchronous FIFO Overview	334
Specifying FIFOs with SYNCORE	335
SYNCORE FIFO Wizard	340
FIFO Read and Write Operations	349
FIFO Ports	350
FIFO Parameters	353
FIFO Status Flags	355
FIFO Programmable Flags	358
SYNCORE RAM Compiler	365
Specifying RAMs with SYNCORE	365
SYNCORE RAM Wizard	373
Single-Port Memories	377
Dual-Port Memories	379
Read/Write Timing Sequences	383
SYNCORE Byte-Enable RAM Compiler	387
Functional Overview	387
Specifying Byte-Enable RAMs with SYNCORE	388
SYNCORE Byte-Enable RAM Wizard	395
Read/Write Timing Sequences	398
Parameter List	401
SYNCORE ROM Compiler	403
Functional Overview	403
Specifying ROMs with SYNCORE	405
SYNCORE ROM Wizard	410
Single-Port Read Operation	414
Dual-Port Read Operation	415

Parameter List	416
Clock Latency	417
SYNCore Adder/Subtractor Compiler	418
Functional Description	418
Specifying Adder/Subtractors with SYNCore	419
SYNCore Adder/Subtractor Wizard	427
Adder	430
Subtractor	433
Dynamic Adder/Subtractor	436
SYNCore Counter Compiler	442
Functional Overview	442
Specifying Counters with SYNCore	443
SYNCore Counter Wizard	449
UP Counter Operation	452
Down Counter Operation	453
Dynamic Counter Operation	453

Chapter 8: Physical Optimization Tools

Design Planner User Interface	458
Design Planner View	458
Physical Analyst User Interface	464
Physical Analyst View	464

Appendix A: Designing with Achronix

Basic Support for Achronix Designs	472
Achronix Speedster7t Device-specific Support	472
Supported Device Families	473
Netlist Format	473
Achronix Components	475
Achronix Asymmetric RAM	475
Achronix LRAM2K_SDPI Inference	485
Packing Output Registers in the LRAM2K_SDPI	489
Achronix BRAM72K_SDPI Inference	490
Achronix ROMs	491
RAM and ROM Initialization for Achronix	495
Achronix MLP primitives Inference	495
Achronix Latch Inference	497
Achronix Register Inference	511
Register Initialization for Achronix	511
Achronix Shift Register Inference	523

Achronix Device Mapping Options	527
Disable I/O Insertion	527
Time Borrowing	527
Compile Point Remapping in Achronix Designs	528
Achronix set_option Command Options	529
Achronix Tcl set_option Command	531
Tcl project Command	533
Achronix Output Files and Forward Annotation	534
Automatic RTL Attribute Propagation	534
Net Attributes in RTL	534
Forward Annotation of RTL Attributes to the Netlist	537
Forward Annotation of RTL Attributes Applied on Pins	538
Integration with Achronix Tools and Flows	540
Hierarchical Area Report	540
Running Post-Synthesis Simulation	540
Designing for Achronix Architectures	541
Achronix Attribute and Directive Summary	542

Appendix B: Designing with Intel FPGA

Basic Support for Intel FPGA Designs	548
Intel FPGA Device Support	548
Netlist Format	549
Supported Synthesis Design Flows for Intel FPGA Designs	550
Inputs for Intel FPGA Designs	552
Intel FPGA Components	554
Intel FPGA RAM and ROM Implementations	557
Intel FPGA DSP Blocks	567
Intel FPGA LPMs	593
Intel FPGA Constraints, Attributes, and Options	599
Intel FPGA I/O Standards	599
Register Packing in Intel FPGA Designs	614
Multi-dimensional Array Support in VQM	615
Intel FPGA Device Mapping Options	616
Fanout Limits in Intel FPGA Designs	617
Compile Point Remapping in Intel FPGA Designs	619
Stratix, Arria, and Cyclone Families	621
MAX II, MAX V, MAX 10 Families	624
MAX Family	628
set_option Command for Intel FPGA Architectures	631

Integration with Intel FPGA Tools and Flows	637
Intel FPGA New Mapper	637
Clearbox Support	638
Greybox Support	640
Working with Intel FPGA PLLs	640
Instantiating Special Buffers as Black Boxes in Intel FPGA Designs	641
Specifying Intel FPGA I/O Locations	643
Packing I/O Cell Registers in Intel FPGA Designs	643
Specifying HardCopy and Stratix Companion Parts	644
Specifying Core Voltage in Stratix III Designs	645
Using LPMs in Simulation Flows	646
 Intel FPGA Output Files and Forward Annotation	649
Intel FPGA Reports	649
Forward Annotating Intel FPGA Output	652
Forward-annotation for Intel FPGA Constraints	653
 Running Post-Synthesis Simulation with VHM Netlist	656
Limitations	656
 Intel FPGA Place-and-Route Guidelines	657
Run Intel FPGA Placement and Routing	657
VQM Filenames for Place-and-Route Tool	658
Quartus II Incremental Compilation Flow	659
 Intel FPGA Attribute and Directive Summary	660

Chapter C: Designing with Lattice

Basic Support for Lattice Designs	666
Supported Device Families	666
Netlist Format	667
 Lattice Features	671
Lattice Radian Software Support	671
Lattice Block RAM Support	672
SYNCORE RAMs	682
Macros and Black Boxes	683
Programmable I/O Cell Latches	683
Lattice iCE RAM and ROM Support	683
Lattice iCE Component Optimizations	684
Initial Value Support for Registers	687
Hierarchical Attribute Support for Module Generation	688
Shift Chain Inference	689
SB_MAC16 Block Inference	690
Lattice Latch Support	693

Timing Propagation Support for Lattice Oscillators	696
Timing Propagation Through PLLs	698
Timing Propagation Support for Instantiated Primitives	702
Lattice Macros and Black Boxes	703
Lattice Constraints, Attributes, and Options	706
Lattice I/O Standards	706
Disable I/O Insertion Globally or Port by Port	707
I/O Buffer Support	707
Global Buffer Promotion for Control Signals	708
Hierarchical Attribute Support for Module Generation	708
Inferring Carry Chains in Lattice XPLD Devices	709
Selective Clock Enable Inference	709
Lattice Device Mapping Options	710
Fanout Guide	711
Disable I/O Insertion	711
GSR Resource	711
Compile Point Remapping in Lattice Designs	712
Write Declared Clocks Only	713
Lattice Device Mapping Options (Older Technologies)	714
ORCA Devices	715
LatticeECP/EC and Later Devices	717
LatticeSC/SCM and LatticeXP2/XP Devices	719
MachXO and Platform Devices	721
ispGAL, ispGDX, and ispLSI Devices	723
ispXPGA Devices	725
ispXPLD5000MX and ispLSI5000VE Devices	726
ispMACH and MACH Devices	728
Lattice iCE40/iCE5LP Devices	730
set_option Command for Lattice Architectures	732
Lattice Design Options	737
FSM Compiler	737
Resource Sharing	737
Pipelining	737
Retiming	738
Gated and Generated Clocks	738
Lattice Output Files and Forward Annotation	740
Synthesis Reports	740
Specifying Pin Locations	741
Specifying Macro and Register Placement	742
Passing Technology Properties	742

Specifying Padtype and Port Information	743
Integration with Lattice Tools and Flows	744
IP Encryption Flow for Lattice	744
Inferring Lattice PIC Latches	744
Controlling I/O Insertion in Lattice Designs	751
Using Lattice GSR Resources	752
Selective Carry-Chain Inferencing	753
Lattice Attribute and Directive Summary	754

Appendix D: Designing with Microchip

Basic Support for Microchip Designs	760
Microchip Device-specific Support	760
Netlist Format	760
Microchip Features	762
Microchip Components	763
Macros and Black Boxes in Microchip Designs	763
DSP Block Inference	765
Control Signals Extraction for Registers (SLE)	771
Wide MUX Inference	771
Microchip RAM Implementations	772
RAM for PolarFire	772
RAM for RTG4	773
RAM for SmartFusion2/IGLOO2	774
PolarFire Asymmetric RAM support	777
Low Power RAM Inference	782
URAM Inference for Sequential Shift Registers	782
Packing of Enable Signal on the Read Address Register	784
Packing of INIT Value on LSRAIM and URAM Blocks in PolarFire	785
RAM Inference in ECC Mode	786
PolarFire RAM Inference for ROM Support	793
Write Byte-Enable Support for RAM	796
RAM Read Enable Extraction (ProASIC3E and Axcelerator)	797
RAM for ProASIC3/3E/3L and IGLOO+/IGLOO/IGLOOe	798
RAM for ProASIC (500K) and ProASICPLUS (PA)	801
RAM for Axcelerator	801
Microchip Constraints and Attributes	802
Microchip I/O Standards	802
Global Buffer Promotion	803
The syn_maxfan Attribute in Microchip Designs	804
Radiation-tolerant Applications	805

Microchip Device Mapping Options	807
Promote Global Buffer Threshold	807
I/O Insertion	808
Retiming	809
Update Compile Point Timing Data Option	809
Operating Condition Device Option	811
Microchip set_option Command Options	813
Microchip Tcl set_option Command Options	814
Microchip Output Files and Forward Annotation	818
VM Flow Support	818
Forward-annotating Constraints for Placement and Routing	820
Specifying Pin Locations	821
Specifying Locations for Microchip Bus Ports	822
Specifying Macro and Register Placement	823
Synthesis Reports	823
Integration with Microchip Tools and Flows	824
Compile Point Synthesis	824
Incremental Synthesis Flow	825
Using Predefined Microchip Black Boxes	825
Using Smartgen Macros	826
Microchip Place-and-Route Tools	827
Microchip Attribute and Directive Summary	828

Appendix E: Designing with Xilinx

Basic Support for Xilinx Designs	834
Xilinx Device Support	834
Netlist Format	834
Xilinx Stacked Silicon Interconnect (SSI) Technology	837
Supported Synthesis Design Flows for Xilinx Designs	838
Xilinx Components	839
DSP Components in Virtex Designs	839
DSP48 Block Inference	840
DSP48 Symmetric Rounding for XNOR Inferencing	847
Dynamic Inmode Support for DSP48 Inferencing	850
Xilinx RAM Inference	854
Xilinx ROM Inference	857
Xilinx Registers	858
Asynchronous Registers	859
DDR Registers	860
Dynamic Shift Register Lookup (SRL) Table	864

Latch Mapping	866
Xilinx I/O Support	866
Macros and Black Boxes	869
Xilinx Constraints, Attributes, and Options	873
Xilinx I/O Standards	873
Relative Locations	877
Wide Adders in Virtex-5 and Virtex-6 Designs	877
Control Sets in Spartan/Virtex Architectures	878
Optimizing Xilinx Designs	879
Designing for Xilinx Architectures	879
Specifying Xilinx Macros	880
Specifying Global Sets/Resets and Startup Blocks	882
Inferring Wide Adders	883
Instantiating CoreGen Cores	886
Instantiating Virtex PCI Cores	887
Initializing Xilinx RAM	889
Specifying Xilinx Register INIT Values	895
Packing Registers for Xilinx I/Os	897
Inserting Xilinx I/Os and Specifying Pin Locations	900
Working with Xilinx Buffers	906
Working with Xilinx Regional Clock Buffers	907
Using Clock Buffers in Virtex Designs	908
Working with Clock Skews in Xilinx Virtex-5 Designs	910
Instantiating Special I/O Standard Buffers for Virtex	911
Specifying RLOCs	912
Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute	914
Xilinx Optimizations	916
Fanout Limits in Xilinx Designs	916
Pipelining in Xilinx Designs	918
Retiming in Xilinx Designs	918
Gated and Generated Clocks in Xilinx Designs	919
Sequential Optimizations in Xilinx Designs	919
Compile Point Remapping in Xilinx Designs	920
Clock Skew Estimation for Xilinx Designs	921
Global Set/Resets and Startup Blocks	921
Advanced LUT Combining	922
Xilinx Device Mapping Options	925
Virtex and Spartan-II and Later Technologies	925
Virtex and Spartan-II and Later Device Mapping Options	925
Virtex and Spartan-II and Later set_option Tcl Command	928

Virtex and Spartan-II and Later project Tcl Command	935
Xilinx CPLDs	935
Xilinx CPLD Device Mapping Options	935
Xilinx CPLD set_option Tcl Command	937
Xilinx CPLD project Command	939
Xilinx Output Files and Forward Annotation	941
Xilinx Resource Usage Reports	941
Xilinx Net Buffering Report	942
Virtex-5 and Virtex-6 Detailed Clock Path Report	942
Custom Timing Reports for Xilinx	943
Forward Annotating Xilinx Output	944
Forward-Annotation of Constraints for Xilinx DSP Inferencing	945
Integration with Xilinx Tools and Flows	947
Reoptimizing with EDIF Files	949
Running Post-Synthesis Simulation	950
Xilinx Attribute and Directive Summary	951

CHAPTER 1

Product Overview

This document is part of a set that includes reference and procedural information for the Synopsys® FPGA synthesis tool. The reference manual provides additional details about the synthesis tool user interface, commands, and features. Use this information to supplement the user guide tasks, procedures, design flows, and result analysis.

The following sections include an introduction to the synthesis tool.

- [Overview of the Synthesis Tools](#), on page 20
- [Synopsys FPGA Tool Features](#), on page 23
- [Graphic User Interface](#), on page 26
- [Getting Help](#), on page 31

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

Overview of the Synthesis Tools

This section introduces the technology, main features, and user interface of the FPGA synthesis tools. See the following for details:

- [Common Features](#), on page 20
- [Synplify Pro and Synplify Premier Features](#), on page 21
- [Synplify Premier Features](#), on page 22
- [Graphic User Interface](#), on page 26

Common Features

The Synopsys FPGA synthesis tool includes the following built-in features:

- The HDL Analyst® analysis and debugging environment, a graphical tool for analysis and crossprobing. See [Analyzing With the HDL Analyst Tool](#), on page 386 and [Analyzing With the Standard HDL Analyst Tool](#), on page 452 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 62.
- The SCOPE® (Synthesis Constraint Optimization Environment®) tool, which provides a spreadsheet-like interface for managing timing constraints and design attributes. See [SCOPE Constraints Editor](#), on page 294.
- FSM Compiler, a symbolic compiler that performs advanced finite state machine (FSM) optimizations. See [Running the FSM Compiler](#), on page 585.
- Integration with the Identify Debugger.

Synplify Pro and Synplify Premier Features

The following features are specific to the Synplify Pro, Synplify Premier, and Synplify Premier with Design Planner tools. For a comparison of the features in the tools, see [Synopsys FPGA Tool Features, on page 23](#).

- FSM Explorer, which tries different state machine optimizations before picking the best implementation. See [Running the FSM Explorer, on page 589](#).
- The FSM Viewer, for viewing state transitions in detail. See [Using the FSM Viewer, on page 405](#).
- The Tcl window, a command line interface for running TCL scripts. See [Tcl Script Window, on page 58](#).
- The Timing Analyst window, which allows you to generate timing schematics and reports for specified paths for point-to-point timing analysis.
- Other special windows, or *views*, for analyzing your design, including the Watch Window and Message Viewer (see [The Project View, on page 34](#)).
- Certain optimizations available, like pipelining and retiming.
- Advanced analysis features like crossprobing and probe point insertion.
- 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 certain technologies (see [Running P&R Automatically after Synthesis, on page 1102](#) in the *User Guide*).

Synplify Premier Features

The following features are specific to the Synplify Premier tool:

- Synthesis Strategy allows you to select a strategy based on fast runtime, routability, or optimal performance for your design.
- Backannotation for place-and-route data. This feature is supported for Intel and Xilinx technologies.
- 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.
- Logic synthesis based on the design plan lets you perform physical optimizations based on the coarse placement. This option is only available for the latest Intel 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 683](#)).
- DesignWare Support - the full Synopsys DesignWare foundation library is supported.

Synopsys FPGA Tool Features

This table distinguishes between major functionality for the Synopsys FPGA products.

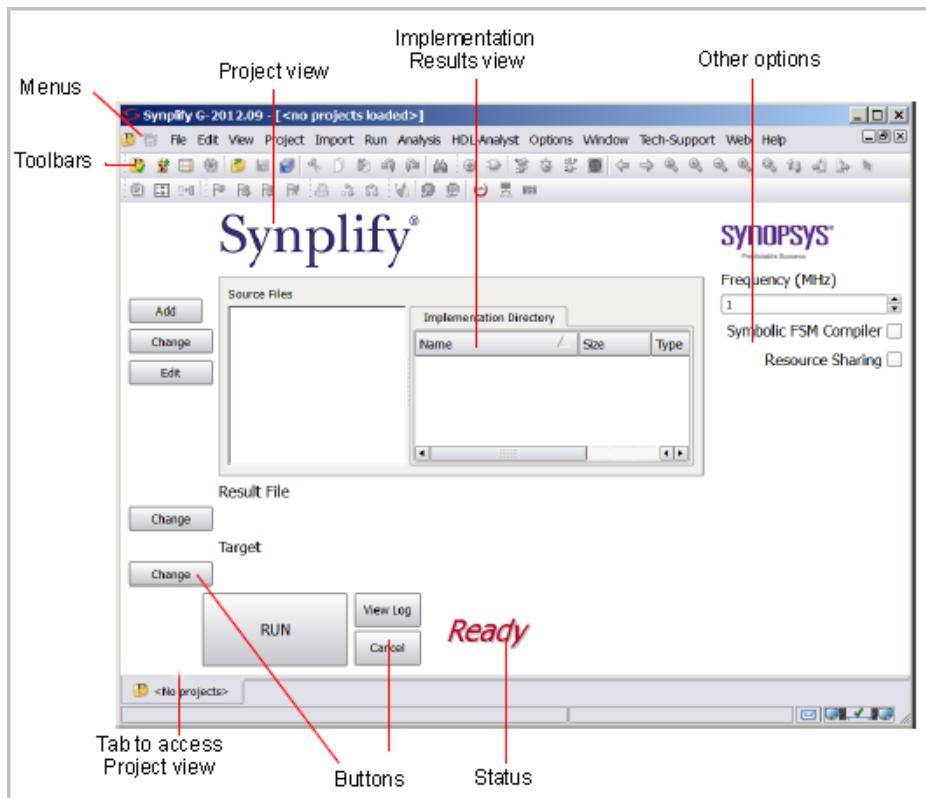
	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Performance				
Behavior Extracting Synthesis Technology® (BEST™)	x	x	x	x
Vendor-Generated Core/IP Support (certain technologies)		x	x	x
FSM Compiler	x	x	x	x
FSM Explorer		x	x	x
Gated Clock Conversion		x	x	x
Register Pipelining		x	x	x
Register Retiming		x	x	x
SCOPE® Constraint Entry	x	x	x	x
High Reliability Features		Limited	x	x
Integrated Place-and-Route	x	x	x	x
Analysis				
HDL Analyst®	Option	x	x	x
Timing Analyzer - Point-to-point		x	x	x
Timing Report View			x	x
FSM Viewer		x	x	x
Crossprobing		x	x	x
Probe Point Creation		x	x	x
Identify® Instrumentor	x	x	x	x
Identify Debugger	x	x	x	x

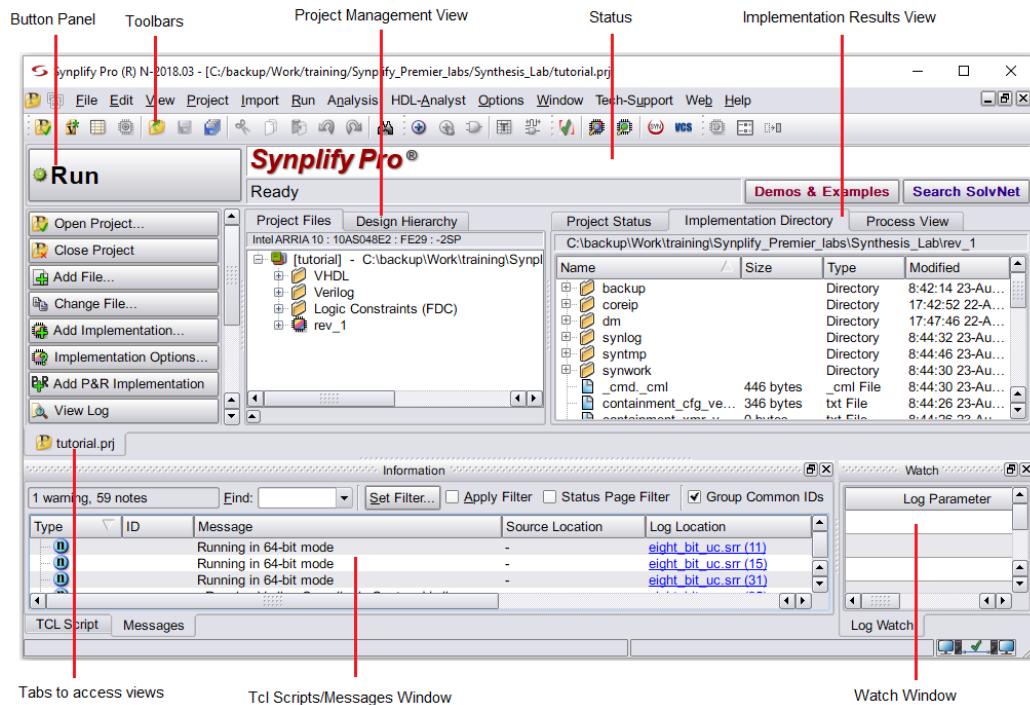
	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Physical Design				
Design Planner				x
Logic Assignment to Regions				x
Area Estimation and Region Capacity				x
Pin Assignment				x
Physical Optimizations			x	x
Physical Analyst			x	x
Synopsys DesignWare® Foundation Library			x	x
Runtime				
Hierarchical Design	x		x	x
Multiprocessing /Distributed Processing			x	x
Compile on Error			x	x
Team Design				
Mixed Language Design	x		x	x
Compile Points	x		x	x
Hierarchical Design	x		x	x
True Batch Mode (Floating licenses only)	x		x	x
GUI Batch Mode (Floating licenses)	x	x	x	x
Batch Mode P&R	-	x	x	x
Back Annotation of P&R Data	-	-	x	x
Identify Integration	Limited	x	x	x
Design Environment				
Text Editor View	x	x	x	x

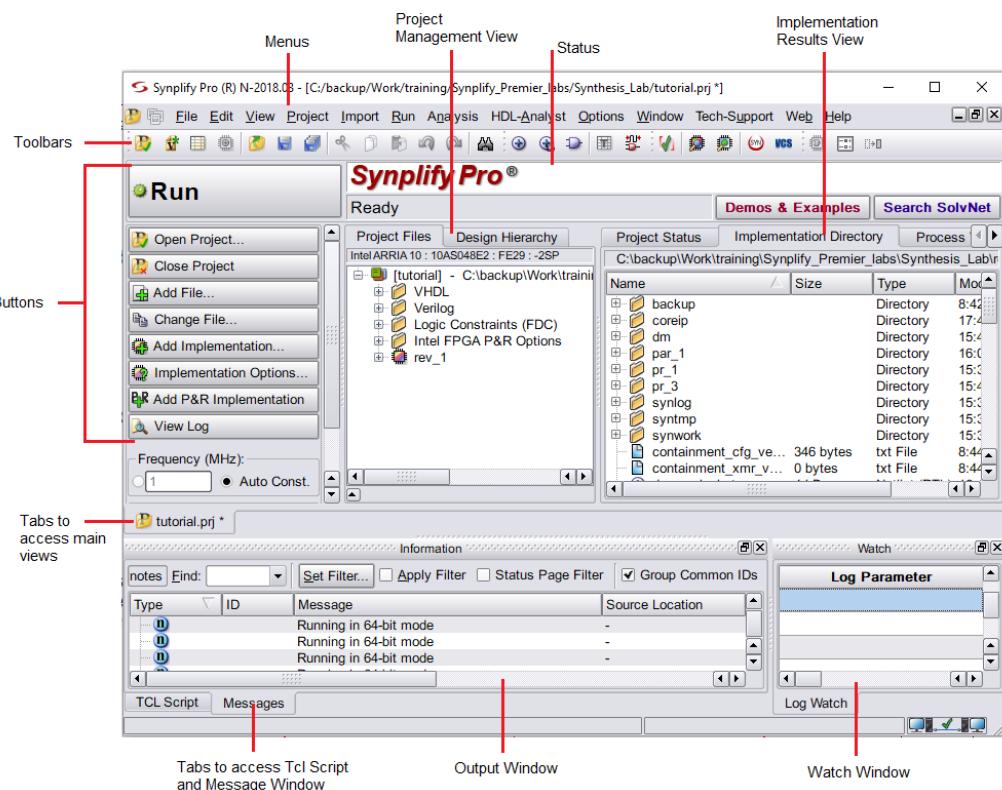
	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Watch Window	x		x	x
Message Window		x	x	x
Tcl Window		x	x	x
Multiple Implementations		x	x	x
Vendor Technology Support	x	x	Selected	Selected
Prototyping Features				
Runtime Features			x	x
Compile Points		x	x	x
Gated Clock Conversion			x	x
Compile on Error			x	x
Unified Power Format (UPF)			x	x

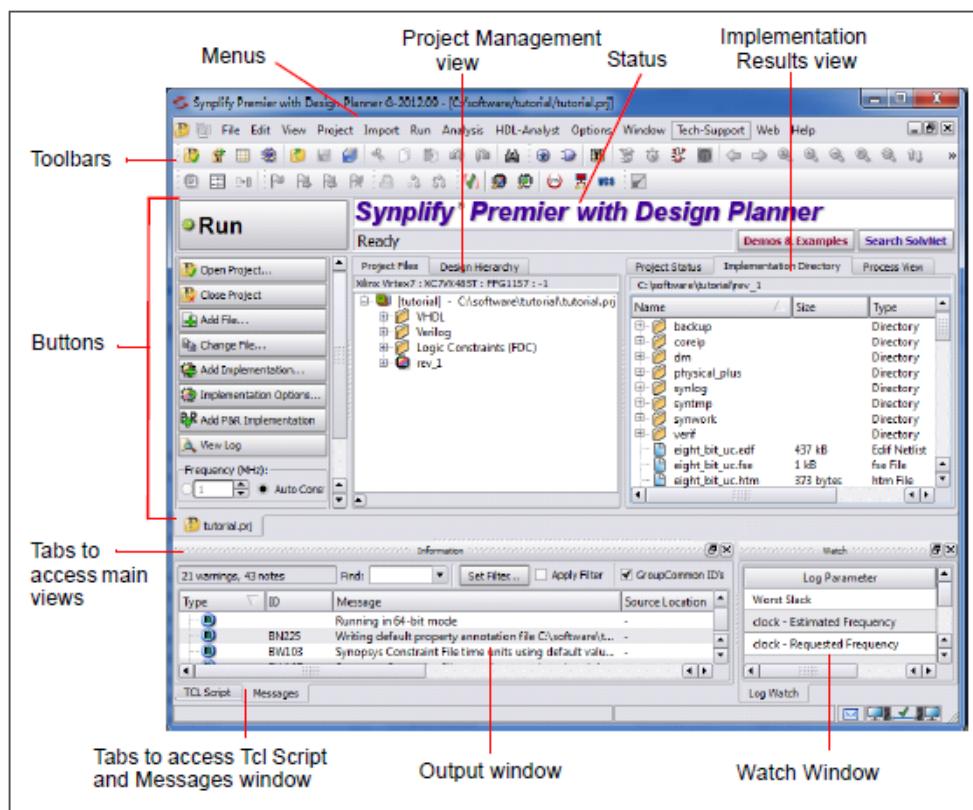
Graphic User Interface

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, with the exception of the Synplify tool. The following figures show the graphical user interfaces for the Synplify, Synplify Pro, and Synplify Premier tools.









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 34
HDL Analyst view	Chapter 7, Analyzing with HDL Analyst
Text Editor view	Text Editor View , on page 62
Tcl window	Tcl Script Window , on page 58
Watch Window	Watch Window , on page 54
SCOPE spreadsheet	SCOPE Constraints Editor , on page 294
Other views and windows	The Project View , on page 34
Menu commands and their dialog boxes	Chapter 5, User Interface Commands
Toolbars	Toolbars , on page 76
Buttons	Buttons and Options , on page 92
Context-sensitive popup menus and their dialog boxes	Chapter 6, GUI Popup Menu Commands
Online help	Use the F1 keyboard shortcut or click the Help button in a dialog box. See Help Menu, on page 602 , for more information.

Getting Help

Look through the documentation for information, before calling Synopsys SolvNetPlus Support. 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...	<i>User Guide</i> and various application notes available on the Synopsys support website
Flow information	<i>User Guide</i> and various application notes available on the Synopsys SolvNetPlus support website
FPGA Implementation Tool	Synopsys Web Page (Web->FPGA Implementation Tools menu command from within the software)
Synthesis features	<i>User Guide</i> and <i>Reference Manual</i>
Language and syntax	<i>Language Support Reference Manual</i>
Attributes and directives	<i>Attribute Reference Manual</i>
Tcl language	Online help (Help->Tcl Help)
Synthesis Tcl commands	<i>Command Reference Manual</i> or type <code>help</code> followed by the command name in the Tcl window
Using tool-specific features and attributes	<i>User Guide</i>
Error and warning messages	Click the message ID code

Document Set

This document is part of a series of books included with the Synopsys FPGA synthesis software tool. The set consists of the following books that are packaged with the tool:

- *FPGA Synthesis User Guide*
- *FPGA Synthesis Reference*

- *FPGA Synthesis Command Reference*
- *FPGA Synthesis Attributes and Directives Reference*
- *FPGA Synthesis Language Support Reference*
- *Identify Instrumentor User Guide*
- *Identify Debugger User Guide*
- *Identify Debugging Environment Reference Manual*

Contacting Customer Support

If you have a problem, question, or enhancement request, contact Synopsys Customer Support.

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 34
- [The Project Results View](#), on page 41
- [Other Windows and Views](#), on page 53
- [Using the Mouse](#), on page 71
- [Toolbars](#), on page 76
- [Keyboard Shortcuts](#), on page 84
- [Buttons and Options](#), on page 92

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 vary for your tool. See the following for an overview:

- [Multiple Pane Project View](#), on page 34
- [Classic Project View](#), on page 36

Multiple Pane Project View

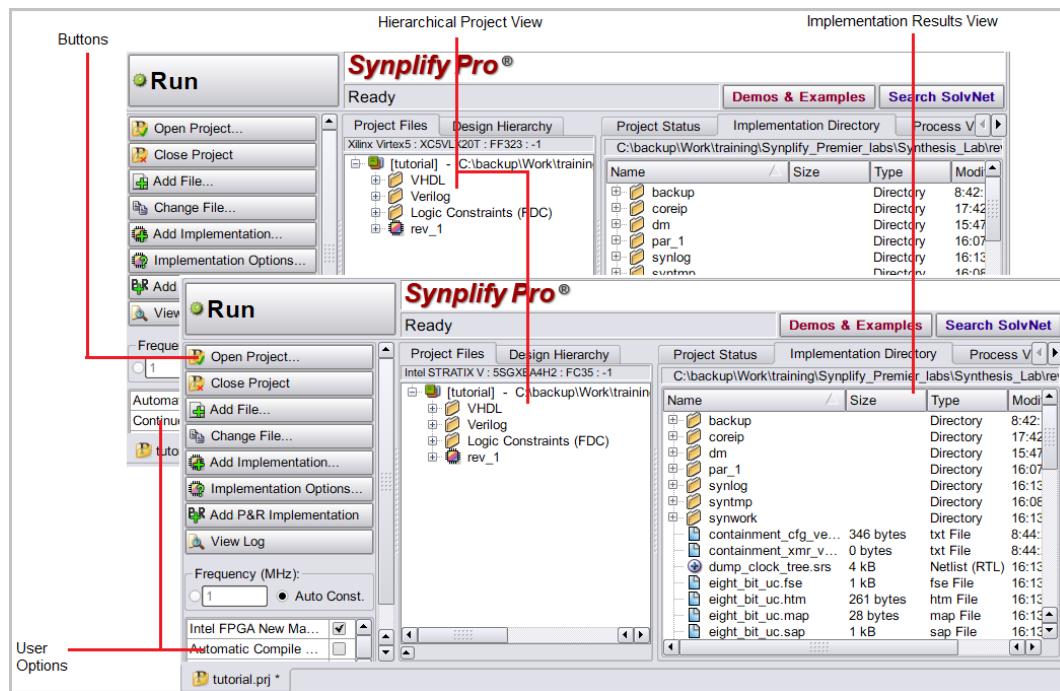
Synplify Pro, Synplify Premier

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.

You can also use the Project Management view to manage and synthesize hierarchical designs.

The following figure shows the main parts of the interface. Additional details about the project view are described here:

- [Project Management View](#), on page 38
- [The Project Results View](#)



The 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.
--------	---

Project View Interface	Description
Buttons and options	Allow immediate access to some of the more common commands. See Buttons and Options , on page 92 for details.
Hierarchical Project Management view	<p>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.</p> <ul style="list-style-type: none"> • Project Files Tab • Design Hierarchy Tab
Implementation Results view	<p>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.</p> <p>The Project Results view includes the following:</p> <ul style="list-style-type: none"> • 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. <p>See The Project Results View , on page 41 for more information.</p>

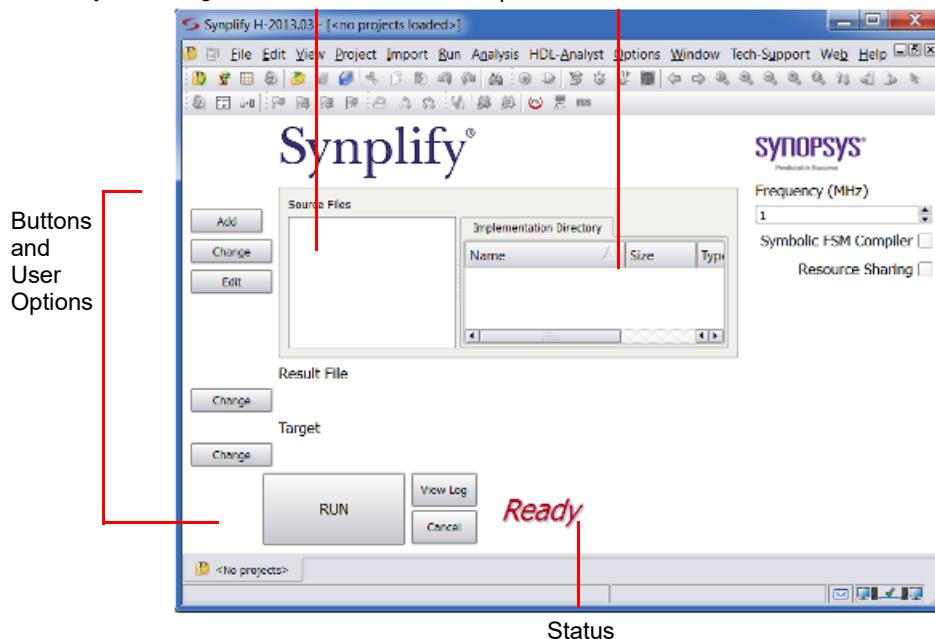
Classic Project View

Synplify

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.

Project Management view

Implementation Results view



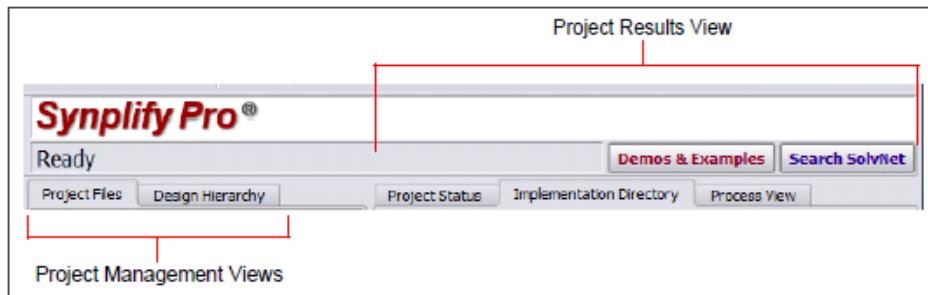
The Project view has the following main parts:

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 505).
Buttons and options	Allow immediate access to some of the more common commands. See Buttons and Options , on page 92 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 33 for details.
Implementation Results view	Lists the result of the synthesis runs for the implementations of your design.

To customize the Project view display, use the Options->Project View Options command ([Project View Options Command, on page 573](#)).

Project Management View

Synplify Pro, Synplify Premier



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 interface.

The synthesis tool provides 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 44](#) in the *User Guide*.

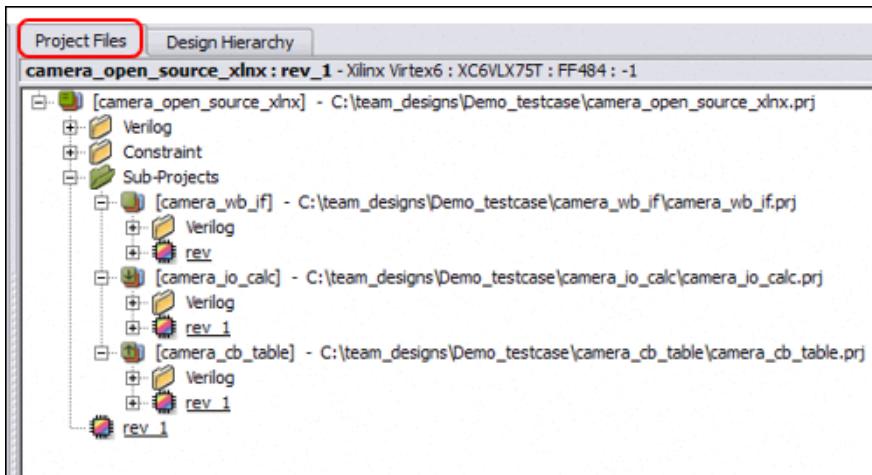
The Project view contains two tabs with different views of the design that help you manage hierarchical projects:

- 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 [Project Management Commands, on page 616](#).

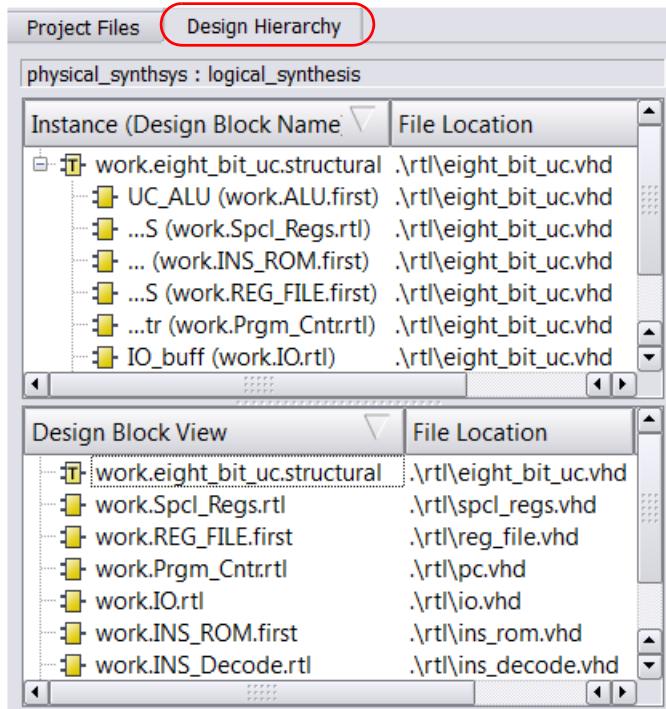
Project Files Tab

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.



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

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](#)
- [Report Tab](#)
- [Timing Report 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 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](#)

For details about how to access synthesis results, see [Accessing Specific Reports Quickly, on page 293](#).

The screenshot shows the Project Status view with the following sections:

- 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	Builds	Warnings	Errors	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 1:28:03 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:**

Optimizations report

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.

The screenshot shows the Project Status view with the Project Settings section expanded. The + icon in the top-left corner of the Project Settings table is circled in red.

Project Name	physical_synthsys	Implementation Name	logical_synthesis
--------------	-------------------	---------------------	-------------------

Project Settings

Project Settings is populated with the project settings from the run_options.txt file after a synthesis run. This section displays information, like the following:

- Project name, top-level module, and implementation name
- Project options currently specified, such as 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 about the synthesis run:

- Job name - Jobs include Compiler Input, Premap, and Map & Optimize. The job might have a Detailed Report link. When you click this link, it takes you to the corresponding report in the log file.

Run Status								
Job Name	Status	①	⚠	❗	CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	22	0	0	-	0m:02s	-	8/8/2013 1:29:29 PM
Premap Detailed report	Complete	4	1	0	0m:00s	0m:00s	78MB	8/8/2013 1:29:30 PM
Map & Optimize Detailed report	Complete	15	1	0	0m:16s	0m:17s	102MB	8/8/2013

Report: tutorial (rev_3)

- ↳ Synthesis
 - ↳ Compiler Report
 - ↳ Pre-mapping Report
 - ↳ Clock Summary
- ↳ Mapper Report
 - ↳ Clock Conversion
 - ↳ Timing Report
 - ↳ Performance Summary
 - ↳ Clock Relationships
 - ↳ Interface Information
 - ↳ Detailed Report for Clocks
 - ↳ Resource Utilization
 - ↳ Hierarchical Area Report(eight bit)
- ↳ Place and Route
 - ↳ Backannotation Report (13:29 08-Aug-2013)
 - ↳ Session Log (13:29 08-Aug-2013)

```
Synopsys Xilinx Technology Pre-mapping. Version map
Copyright (C) 1991-2013, Synopsys, Inc. This software
Product Version 1-2013.09 beta

Mapper Startup Complete (Real Time elapsed 0h:00m:00s)
Linked File: eight_bit.uc.socck.rpt
Printing clock summary report in "C:\sw\tutorial\"
@N:MF245 : | Running in 32-bit mode.
@N:MF566 : | Clock conversion enabled

Design Input Complete (Real Time elapsed 0h:00m:00s)

Mapper Initialization Complete (Real Time elapsed 0h:00m:00s)

Start loading timing files (Real Time elapsed 0h:00m:00s)
routetable is 1.6,1.5999994618778257,1.871712156657
```

- Status - Reports whether the job is running or completed.

- Notes, Warnings, and Errors - These columns are headed by the respective icons and display the number of messages. The messages themselves are displayed in the Messages tab, beside the TCL Script tab. Links are available to the error message and the log location.

Type	ID	Message	Source Location	Log Location	Time	Report
W	MT205	Auto Constraint mode is enabled		eight_bit_hex...	12:29:41...	Pre-mapping Report
W	MR000	Clock conversion enabled		eight_bit_hex...	16:13:34...	Pre-mapping Report
W	FXR04	Found address in waveform eight_bit_hex(verilog) ... aluv(0x)	aluv(0x)	eight_bit_hex...	12:29:48...	Pre-mapping Report
		Found counter in waveform prgnt_01(verilog) ... pcu(13)	pcu(13)	eight_bit_hex...	12:29:48...	Pre-mapping Report

The message numbers may not match for designs with compile points. The numbers reflect the top-level design.

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

Reports

The mapper summary table generates various reports such as an

- Area Summary
- Optimization Summary
- Compile Point Summary
- High Reliability Summary

Click the Detailed Report link when applicable, to go to the log file and information about the selected report. These reports are written to the synlog folder for the active implementation.

Area Summary

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

Run Status

Job Name	Status	W	A	P	CPU Time	Real Time	Memory	Date/Time
Compile Input Data flow report	Complete	64	0	0	-	0m:18s	-	11/4/2011 8:13:36 AM
Promap Data flow report	Complete	5	1	0	0m:13s	0m:13s	1/6MB	11/4/2011 8:12:50 AM
Map & Optimize Netlist report	Complete	334	1143	0	06m:11s	16m:33s	1383MB	11/4/2011 8:28:23 AM

Area Summary

I/O ports	54	Non I/O Register bits	44412 (361%)
I/O Register bits	0	Block RAMs	48 (48)
DSP48s	12		

Detailed report

- Clock Name**
 - obj_aquaris_top|chip_clock
 - obj_aquaris_top|system_clock
- Generated Clock Optimization**

Report: tutorial (rev_3)

- Synthesis**
 - Compiler Report
 - Pre-mapping Report
 - Mapper Report
 - Clock Summary
 - Timing Report
 - Performance Summary
 - Clock Relationships
 - Interface Information
 - Detailed Report for Clocks
 - Resource Utilization
 - Hierarchical Area Report(eight_bit)
- Place and Route
 - Backannotation Report (13:29 08-Aug)
 - Session Log (13:29 08-Aug)

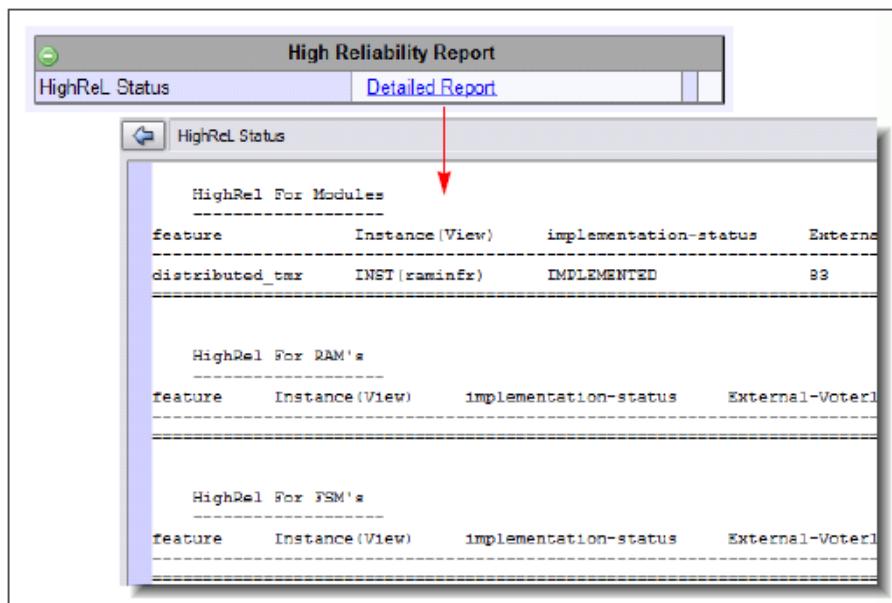
Resource Usage Report For eight_bit_pc

```

Mapping to part: xc7vn33Ct256I167-1i
Cell usage:
DOP4OE1      1 uses
FD            8 uses
FDC           133 uses
FDC2          124 uses
FDE           5 uses
FDD           2 uses
FDPE          24 uses
GND           10 uses
MIXCY_T       18 uses
MUX3T         2 uses
RAM32X25      4 uses
VCC           10 uses
XORCY         20 uses
LUT1           20 uses
LUT2           29 uses
LUT3           9 uses
LUT4           17 uses
LUT5           73 uses
LUT6           154 uses
LUT5_2         2 uses
  
```

High Reliability Report

Click Detailed Report in the High Reliability Report section of the Project Status view to view the high reliability status report. The High Reliability Report is displayed only if one of the High Reliability features (DTMR, DWC, RAM TMR, ECC, Safe FSMs) is being used.



Report Tab

Some reporting such as the Hierarchical Area Report are written to the Report tab of the Project Results view. These reports are typically not included in the log file; therefore, they are displayed separately.

Hierarchical Area Report

The hierarchical area report is supported for the following technology families.

Vendors	Technologies
Achronix	<ul style="list-style-type: none">Speedster22HD family
Intel	<ul style="list-style-type: none">Arria GX and newer familiesCyclone and newer familiesHardcopy II and newer familiesStratix and newer families
Microchip	<ul style="list-style-type: none">IGLOO2 familyRTG4 familySmartfusion2 family
Xilinx	<ul style="list-style-type: none">Artix-7 familiesKintex-7 familiesSpartan-6 familiesUltraScale and UltraScale+ FPGA familiesVirtex-5 and newer familiesZynq families

A Hierarchical Area report is generated in a Report tab 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.

Use the arrow icon () to get back to the main Project Status view.

Project Settings

Project Name	tutorial	Implementation Name	rev_1
--------------	----------	---------------------	-------

Run Status

Job Name	Status	W	A	B	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: 0s	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 Summary

Module name	LUTS	REGISTERS	SYNC RAMS	DSP48
eight_bit uc	265	253	1	1
	21	1	1	0
	38	10	0	1
	17	16	0	0
	33	18	0	0
	37	16	0	0
	84	106	0	0
	0	5	0	0
	3	73	0	0

Timing Report View

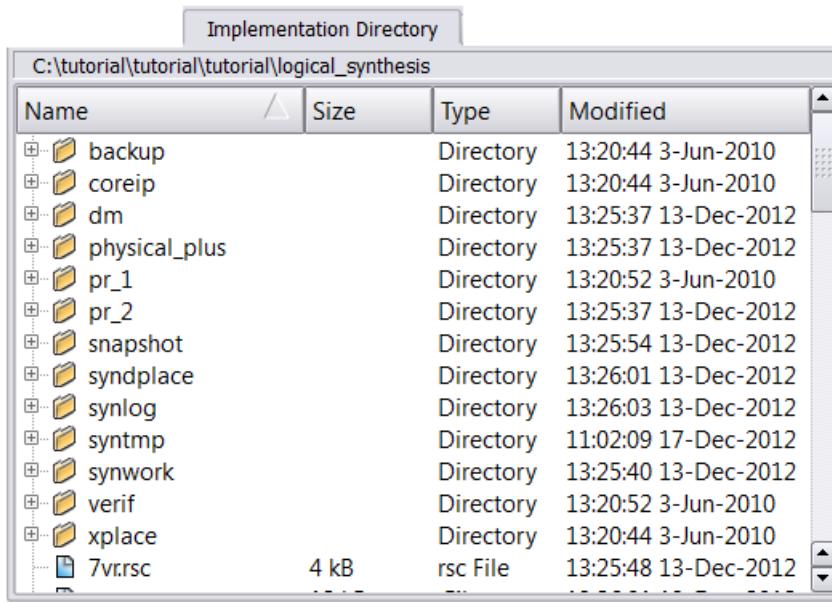
Synplify Premier

The Timing Report View displays the synthesis timing report summary and the P&R timing and correlation report. You can also access this report by clicking on **Timing Report View** from the Project Status view. For details about this timing report, see [Using the Timing Report View, on page 497](#) and [Timing Report View, on page 520](#).

Timing Summary			
Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
raminfr_top clk	1.0 MHz	NA	NA
Detailed report		Timing Report View	

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.

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.

Process View				
Process	State	Run/File Time	TCL Name	
logical_synthesis	<input checked="" type="checkbox"/> Show Hierarchy			
Logic Synthesis	Running.	00:00:00	synthesis	
Compile	out-of-date	00:00:00	compile	
Compile Process	Complete	00:00:00	compile_flow	
Premap	out-of-date	00:00:00	premap	
Map	Complete	00:00:00	map	
Map & Optimize	Complete	00:00:00	fpga_mapper	
Place & Route (pr_1)	Running.	00:00:00	pr_1	
Xilinx P&R (xtclsh)	out-of-date	00:00:00	xilinx_xtclsh	

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, and map. 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 to run jobs at any stage of the design process, for example map. Right-click 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.

Other Windows and Views

Besides the Project view, the tool provides 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 54
- [Watch Window](#), on page 54
- [Tcl Script and Messages Windows](#), on page 57
- [Tcl Script Window](#), on page 58
- [Message Viewer](#), on page 58
- [Output Windows \(Tcl Script and Watch Windows\)](#), on page 62
- [Text Editor View](#), on page 62
- [Context Help Editor Window](#), on page 65
- [Demos & Examples](#), on page 67
- [Interactive Attribute Examples](#), on page 68
- [Search SolvNetPlus](#), on page 70

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

Project View

[The Project View](#), on page 34

SCOPE Interface

[SCOPE Tabs](#), on page 295

HDL Analyst Schematic

[Chapter 7, Analyzing with HDL Analyst](#)

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

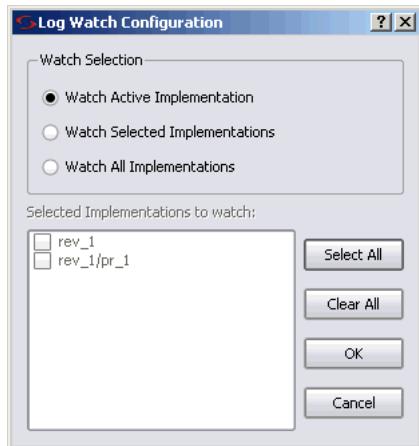
Synplify Pro, Synplify Premier

The Watch window displays selected information from the log file (see [Log File, on page 199](#)) 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.



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.

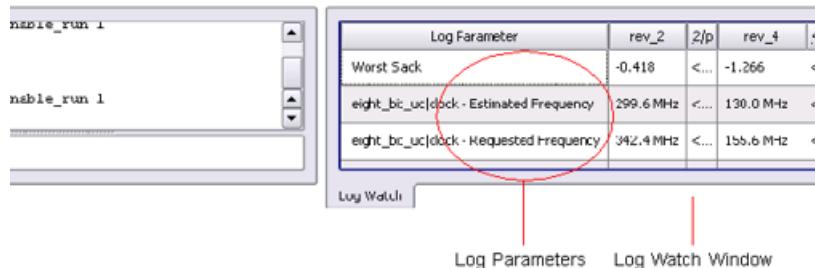
The Watch window has a special positioning popup menu that you access by right-clicking the window border. The following commands are in the menu:

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).

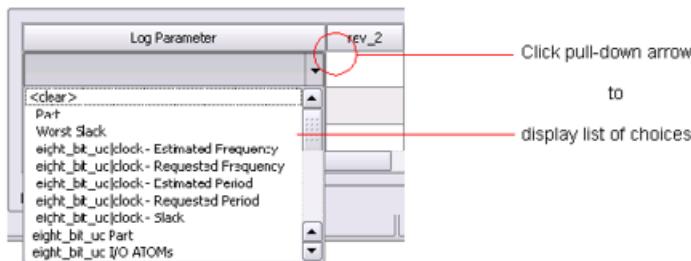
Right-clicking the window *title bar* when the Watch window is floating displays an alternative popup menu with commands Hide and Move; Move lets you position the window using either the arrow keys or the mouse.

Using the Watch Window

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



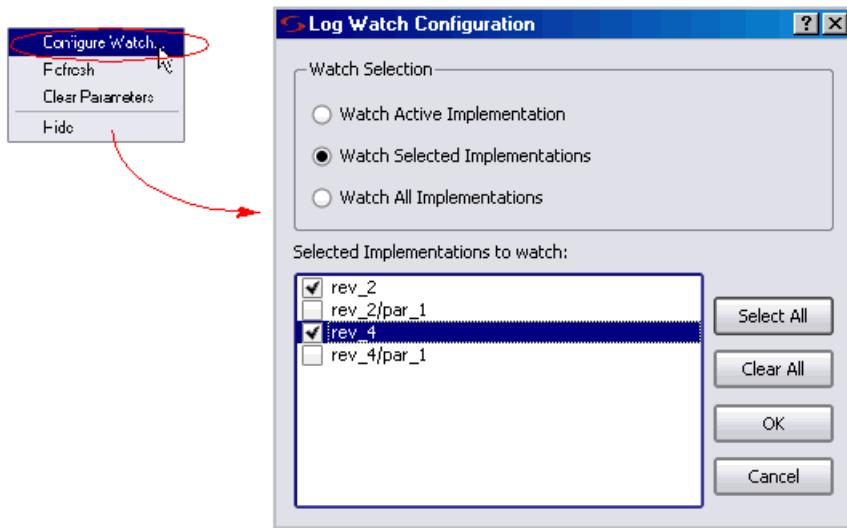
To choose log parameters from a pull-down menu, click in the Log Parameter section of the window. Click the pull-down arrow that appears to display the parameter list choices:



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.



Tcl Script and Messages Windows

Synplify Pro, Synplify Premier

The Tcl 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 58](#) and [Message Viewer, on page 58](#).

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.

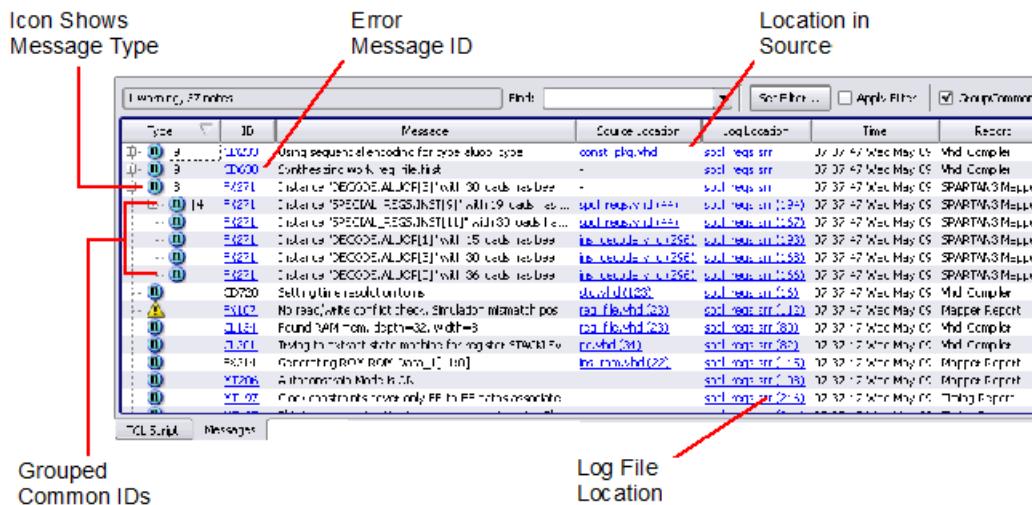
The Tcl script window also displays each command executed in the course of running the synthesis tool, regardless of whether it was initiated from a menu, button, or keyboard shortcut. Right-clicking inside the Tcl window displays a popup menu with the Copy, Paste, Hide, and Help commands.

See also

- [Tcl Synthesis Commands, on page 23](#), for information about the Tcl synthesis commands.
- [Generating a Job Script, on page 782](#) 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.



Interactive tasks in the Messages panel include:

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

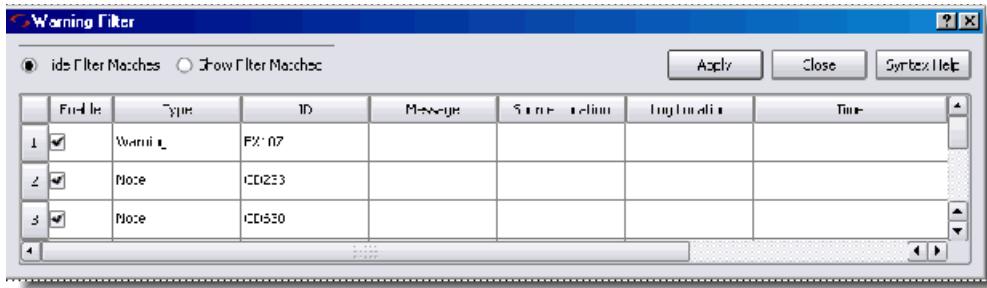
The following table describes the contents of the Messages panel. You can sort the messages by clicking the column headers. For further sorting, use Find and Filter. For details about using this window, see [Checking Results in the Message Viewer, on page 306](#) in the *User Guide*.

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 61.
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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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 that 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.

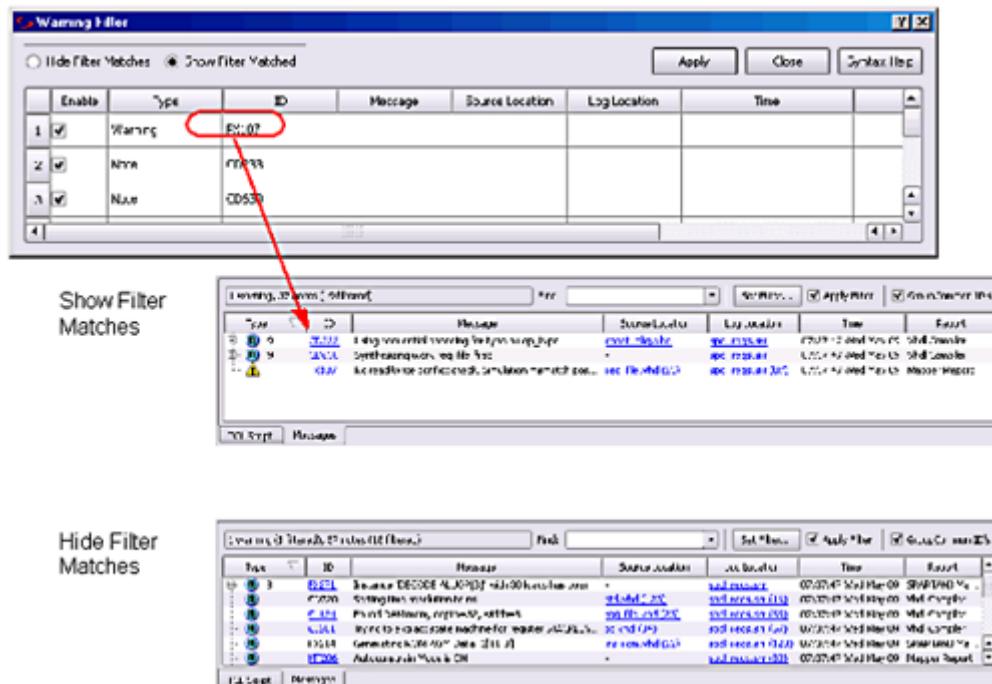
Messages Filter

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 306](#) 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.



Output Windows (Tcl Script and Watch Windows)

The Output windows are the Tcl Script and Log Watch windows. To display or hide them, use View->Output Windows from the main menu. Refer to [Watch Window, on page 54](#) and [Tcl Script and Messages Windows, on page 57](#) for more information.

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 72](#) in the *User Guide*.

```

12  CLK : in std_logic;
13  RESET: in std_logic;
14  INST : in std_logic_vector(11 downto 0);
15  LONGK :out std_logic_vector(8 downto 0);
16  ALUOP : out ALUOP_TYPE;
17  FWE : out std_logic;
18  W_Reg_Write : out std_logic;
19  ALUA_SEL : out ALU_SEL_TYPE;
20  ALUB_SEL : out ALU_SEL_TYPE;
21  STATUS_Z_WRITE : out std_logic;
22  STATUS_C_WRITE : out std_logic;
23  TRIS_WE : out std_logic;
24  TWO_CYC_INST : out std_logic;
25  SKIP_INST : out std_logic;
26  OPCODE_GOTO : out std_logic;
27  OPCODE_CALL : out std_logic;
28  OPCODE_RETlw : out std_logic
29 );
30 end INS_Decode;
31
32 Architecture RTL of Ins_Decode is
33

```

Ln | 3 Col | 14 Total | 331 | Ovr | Block

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 .vhdl) in the Project view.
- Choose File ->Open. In the dialog box displayed, double-click a file to open it.

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

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

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

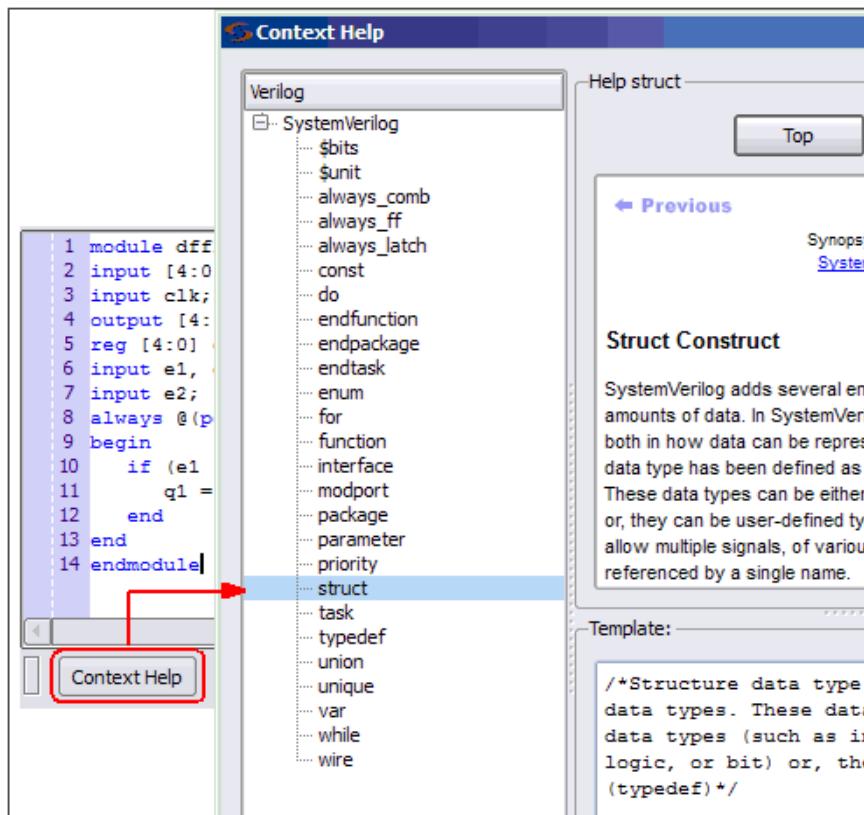
Feature	Description
Color coding	Keywords are blue, comments green, and strings red. All other text is black.
Editing text	You can use the Edit menu or keyboard shortcuts for basic editing operations like Cut, Copy, Paste, Find, Replace, and Goto.
Completing keywords	To complete a keyword, type enough 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.
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 580](#), for information on setting Text Editor preferences.
- [File Menu, on page 396](#), for information on printing setup operations.
- [Edit Menu Commands for the Text Editor, on page 403](#), for information on Text Editor editing commands.
- [Text Editor Popup Menu, on page 607](#), for information on the Text Editor popup menu.
- [Text Editor Toolbar, on page 80](#), for information on bookmark icons of the Edit toolbar.
- [Keyboard Shortcuts, on page 84](#), for information on keyboard shortcuts that can be used in the Text Editor.

Context Help Editor Window

Use the Context Help button to copy Verilog, SystemVerilog, or VHDL constructs into your source file or Tcl constraint commands into your Tcl file. When you load a Verilog/SystemVerilog/VHDL file or Tcl file into the UI, the Context Help button displays at the bottom of the window. Click this button to display the Context Help Editor.



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 or command 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 or command shown in the template window is inserted into your file at the location of the cursor. This allows you to easily insert the code or constraint command 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 or constraint command 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.

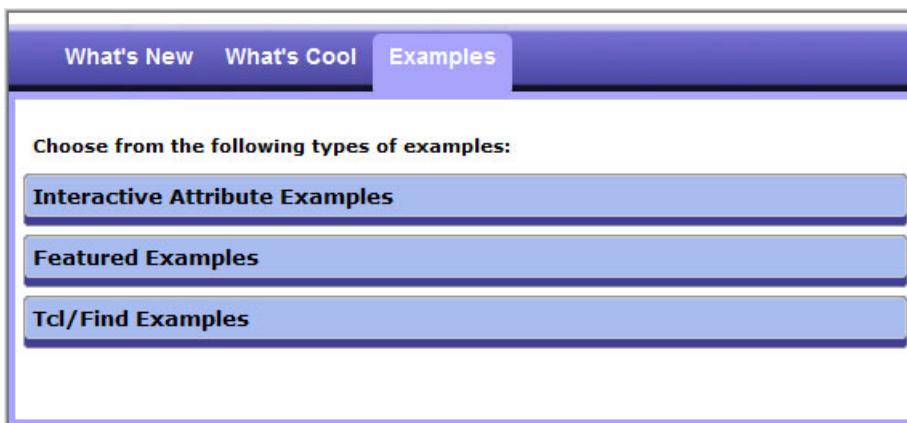
Demos & Examples

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 and links to videos, online help sections and SolvNetPlus articles. 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 68.

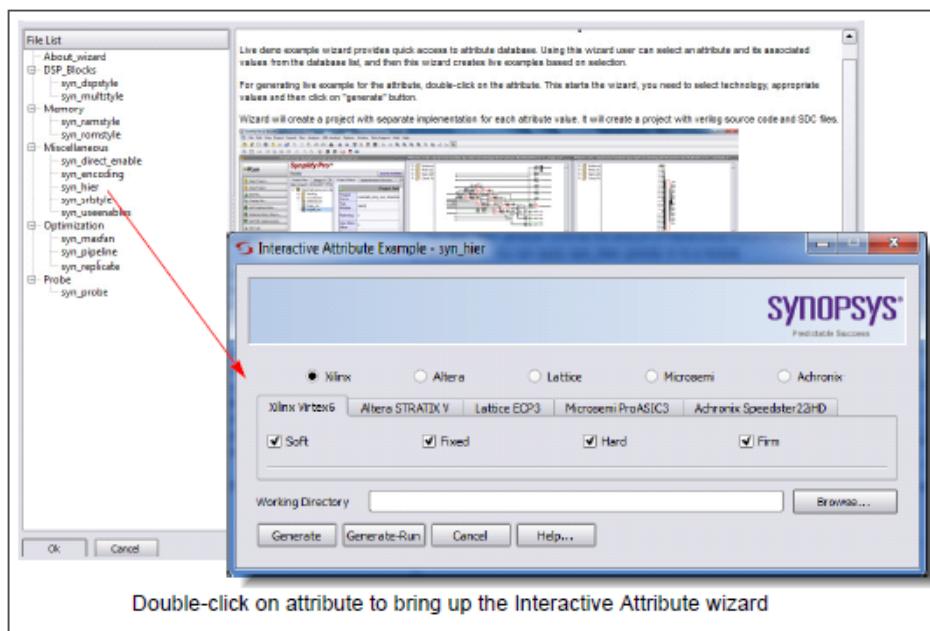
Note: When running code examples, make sure to change the Implementation Directory from the installation directory to your own results directory.



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 click Interactive Attribute Examples and the Launch Interactive Attributes Wizard links.



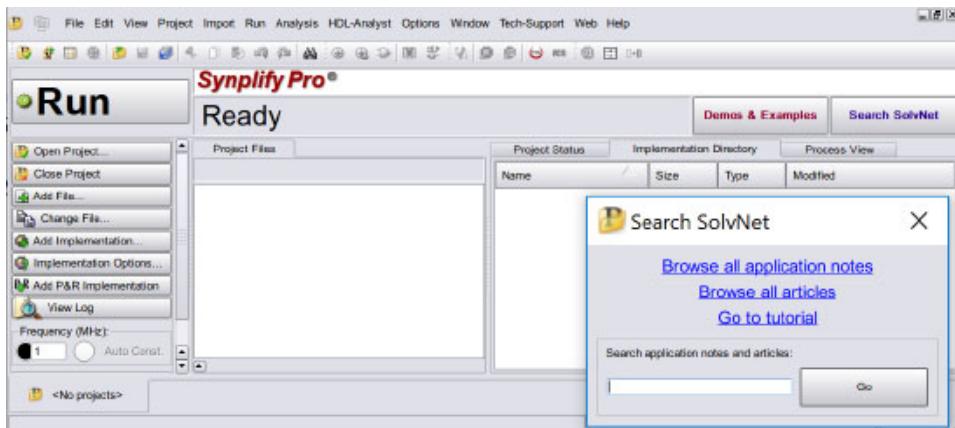
3. Double-click an attribute to start the wizard.
4. Specify the Working Directory location to write your project.
5. Click Generate to generate a project for your attribute.

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 SolvNetPlus

The Synopsys FPGA synthesis tool provides an easy way to access SolvNetPlus from within the Project view. Click 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:

- 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 the Browse all application notes link—takes you to a SolvNet page that links to all the Synopsys FPGA products Application Notes.
- Click the Browse all articles link—takes you to the Browse Articles by Product SolvNet page.
- Click the Go to tutorial link—takes you to the tutorial page for the Synopsys FPGA product you are using (same as Help->Tutorial).

Using the Mouse

The mouse button operations in Synopsys FPGA product is standard; refer to [Mouse Operation Terminology](#) for a summary of supported functions. The tool provides support for:

- [Using Mouse Strokes](#), on page 72
- [Using the Mouse Buttons](#), on page 73
- [Using the Mouse Wheel](#), on page 75

Mouse Operation Terminology

The following terminology is used to refer to mouse operations:

Term	Meaning
Click	Click with the <i>left</i> 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 “ <i>dropping</i> ”. 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.

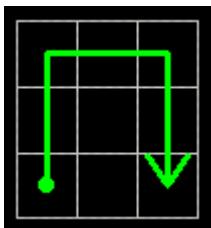
Using Mouse Strokes

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).

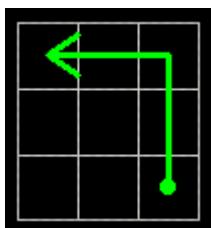
Some strokes are context sensitive. That is, the interpretation of the stroke depends upon the window in which the stroke is started. For example, in an HDL Analyst view, the right stroke means “Next Sheet.” In a dialog box, the right stroke means “OK.”

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

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

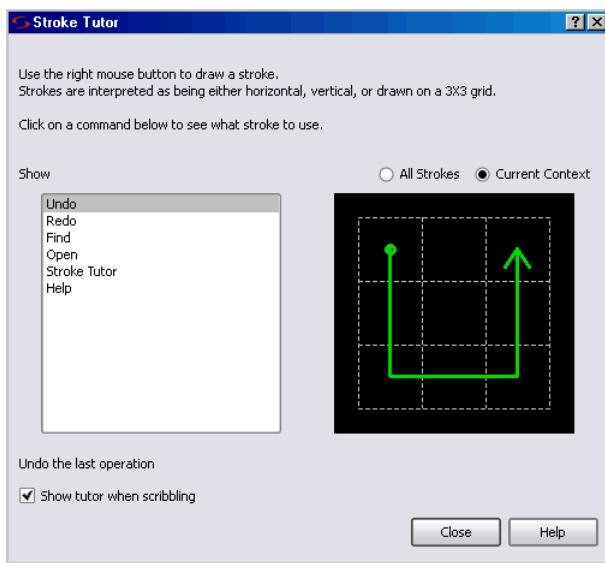


Back to Previous View

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)



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 HDL Analyst window, the Current Context option in the tutor shows only those strokes recognized in the HDL Analyst window.

You can display the tutor while working in a window such as the HDL Analyst 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 [Working in the Schematic, on page 338](#) of the *FPGA Synthesis User Guide*, 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.

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.

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 compiled and mapped schematic 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.
- [Tools Toolbar](#) — Opens supporting tool.

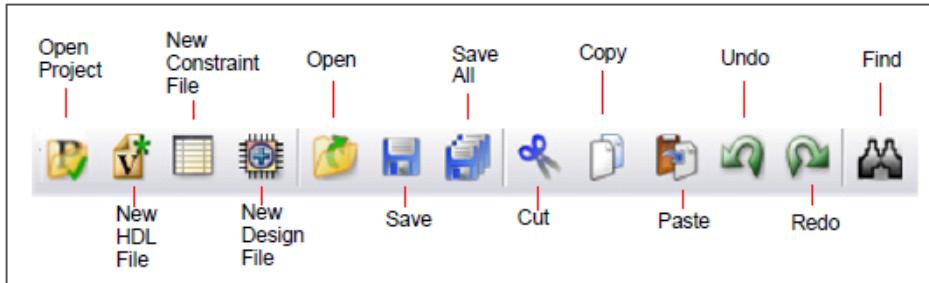
You can enable or disable the display of individual toolbars - see [Toolbar Command, on page 418](#).

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:

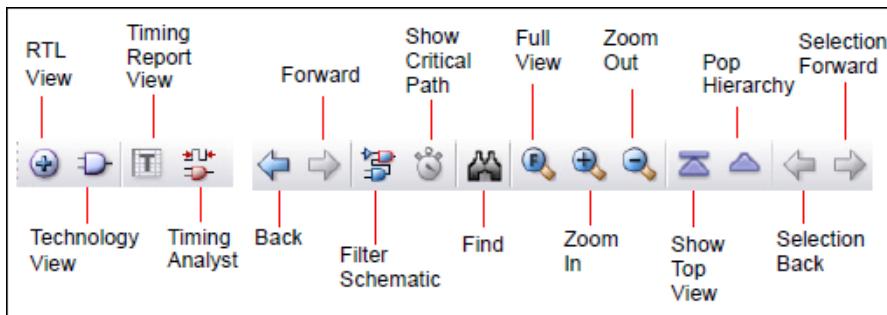
- [File Menu, on page 396](#)
- [Edit Menu, on page 402](#)

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 <i>Synplify Premier</i> 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.

Icon	Description
	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.
	Saves all files associated with the current design. Same as File ->Save All.
	Cuts text or graphics from the active view, making it available to Paste. Same as Edit ->Cut.
	Pastes previously cut or copied text or graphics to the active view. Same as Edit ->Paste.
	Undoes the last action taken. Same as Edit ->Undo.
	Performs the action undone by Undo. Same as Edit ->Redo.
	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 542](#), for more information.

Icon	Description
 RTL View	<p>Opens a new, hierarchical RTL view: a register transfer-level schematic of the compiled design, together with the associated Hierarchy Browser.</p> <p>Same as HDL Analyst ->RTL ->Hierarchical View.</p>
 Technology View	<p>Opens a new, hierarchical Technology view: a technology-level schematic of the mapped (synthesized) design, together with the associated Hierarchy Browser.</p> <p>Same as HDL Analyst ->Technology ->Hierarchical View.</p>
 Timing Report View	<p><i>Synplify Premier</i></p> <p>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.</p> <p>Same as Analysis ->Timing Report View.</p> <p>Same as Project Status -> Timing Summary ->Timing Report View.</p>
 Timing Analyst	<p>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 519.</p> <p>Only available for certain device technologies.</p> <p>Same as Analysis ->Timing Analyst.</p>
 Filter Schematic	<p>Filters your entire design to show only the selected objects. The result is a <i>filtered</i> schematic.</p> <p>Same as HDL Analyst ->Filter Schematic.</p>

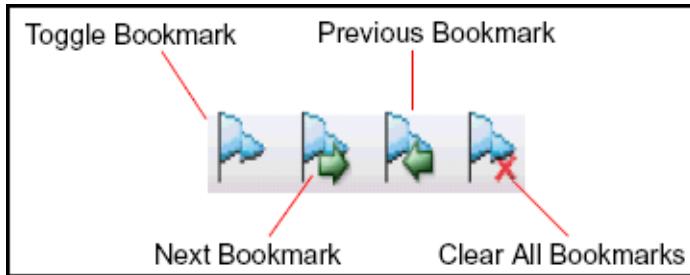
Icon	Description
 Show Critical Path	Filters your design to show only the instances (and their paths) whose slack times are within the slack margin of the worst slack time of the design (see HDL Analyst ->Set Slack Margin). The result is flat if the entire design was already flat. Available only in a Technology view. Not available in a Timing view.
 Back	Goes backward in displaying schematics of the current HDL Analyst view. Same as View ->Back.
 Forward	Goes forward in displaying schematics of the current HDL Analyst view. Same as View ->Forward.
 Zoom In	Zooms the view in or out. Buttons stay active until deselected.
 Zoom Out	Same as View ->Zoom In or View ->Zoom Out.
 Zoom Full	Zoom that reduces the active view to display the entire design. Same as View ->Full View.
 Show Top Level	Displays the schematic for the top-level view.
 Pop Hierarchy	Traverses the schematic hierarchy using pop mode.
 Selection Back	Displays the previous schematic that was selected.
 Selection Forward	Toggles back to the original schematic that was previously selected.

Physical Analyst Toolbar

For details, see [Physical Analyst Toolbar, on page 683](#).

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 77](#).) The Edit toolbar provides the following icons, by default:



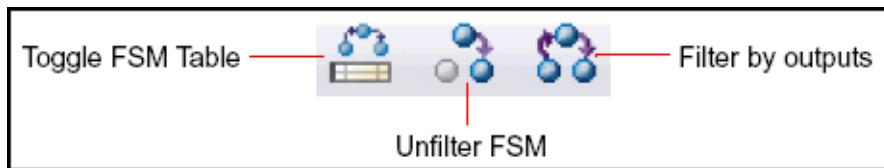
The following table describes the default Edit icons. Each is available in the Text Editor, and each is equivalent to an Edit menu command there - see [Edit Menu Commands for the Text Editor, on page 403](#), for more information.

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

Synplify Pro, Synplify Premier

When you push down into a state machine primitive in an RTL view, the FSM Viewer displays and enables the FSM toolbar. 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 415](#), 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 tool.

Icon	Description
	Constraint Check Checks the syntax and applicability of the timing constraints in the constraint file for your project and generates a report (<i>project_name_cck.rpt</i>). Same as Run->Constraint Check.
	Identify Instrumentor Brings up the Synopsys Identify Instrumentor product. For more information, see Working with the Identify Tools, on page 1159 of the User Guide.
	Launch Identify Debugger Launches the Synopsys Identify Debugger product. For more information, see Working with the Identify Tools, on page 1159 of the User Guide.
	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 508 .
	VCS Simulator Configures and launches the VCS simulator.

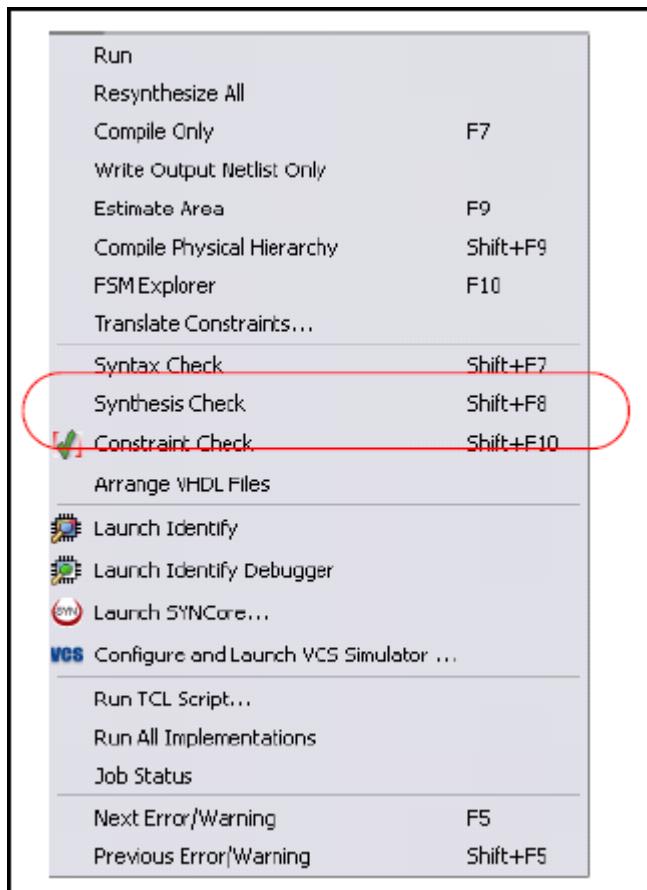
Design Planner Toolbar

For details, see [Design Planner UI Commands, on page 658](#).

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 tool keyboard shortcuts, see [Design Planner Keyboard Shortcuts, on page 681](#) and [Physical Analyst Keyboard Shortcuts, on page 684](#).

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.

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 <i>filtered</i> schematic. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Expand Paths (see HDL Analyst Menu: Filtering and Flattening Commands, on page 546).
Ctrl-++ (number pad)	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)	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)	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 <i>filtered</i> 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 544).
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 407 .
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 544).

Keyboard Shortcut	Description
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 <i>filtered</i> 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 544).
Ctrl-Enter (Return)	In the FSM Viewer, hides all but the selected state(s).
	Same as View->Filter->Selected (see View Menu, on page 415).
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 403).
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 403).
	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 416).
Ctrl-h	In the Text Editor, replaces text. Same as Edit->Replace (see Edit Menu Commands for the Text Editor, on page 403).
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 544).
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 544).

Keyboard Shortcut	Description
Ctrl-l	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 415).
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 544).
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 <i>filtered</i> 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 544).
Ctrl-s	In the Project View, saves the file. Same as File ->Save.
Ctrl-t	Toggles display of the Tcl window. Same as View ->Tcl Window (see View Menu, on page 415).
Ctrl-u	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 403). 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 417).
Ctrl-v	Pastes the last object copied or cut. Same as Edit ->Paste.

Keyboard Shortcut	Description
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 416). 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 416). 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 403).
Ctrl-Shift-h	In an RTL or Technology view, shows all pins on selected <i>transparent</i> 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 549).
Ctrl-Shift-i	In an RTL or Technology view, selects all instances on the current schematic level (all sheets). This does <i>not</i> select instances on other levels. Same as HDL Analyst->Select All Schematic->Instances (see HDL Analyst Menu, on page 542).
Ctrl-Shift-p	In an RTL or Technology view, selects all ports on the current schematic level (all sheets). This does <i>not</i> select ports on other levels. Same as HDL Analyst->Select All Schematic->Ports (see HDL Analyst Menu, on page 542).
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 403).

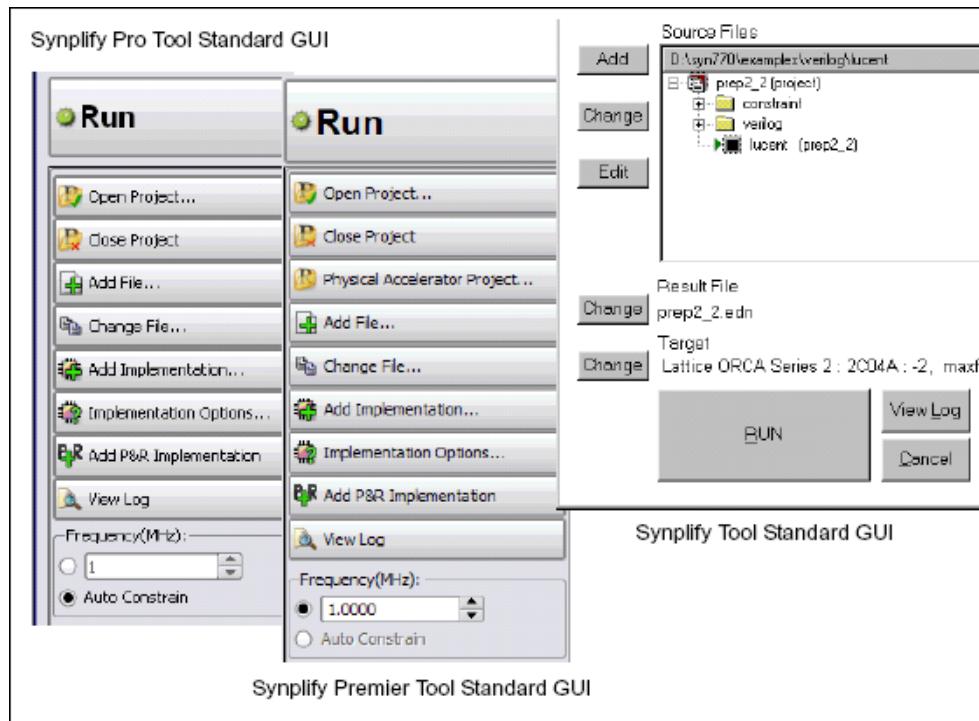
Keyboard Shortcut	Description
d	In an RTL or Technology view, selects the driver for the selected net. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Select Net Driver (see HDL Analyst Menu, on page 542).
Delete (DEL)	Removes the selected files from the project. Same as Project->Remove 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 542).
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 416). 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 403).
F4	In the Project view, adds a file to the project. Same as Project->Add Source File (see Build Project Command, on page 401). 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 416).
F5	Displays the next source file error. Same as Run->Next Error/Warning (see Run Menu, on page 498).
F7	Compiles your design, without mapping it. Same as Run->Compile Only (see Run Menu, on page 498).
F8	Synthesizes (compiles and maps) your design. Same as Run->Synthesize (see Run Menu, on page 498).
F11	Toggles zooming in. Same as View->Zoom In (see View Menu: RTL and Technology Views Commands, on page 416).

Keyboard Shortcut	Description
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 546 .
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 542).
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 542).
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 <i>filtered</i> schematic. Limited to the current schematic. Same as HDL Analyst ->Current Level->Expand to Register/Port (see HDL Analyst Menu, on page 542).
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).
Shift-F5	Displays the previous source file error. Same as Run->Previous Error/Warning (see Run Menu, on page 498).
Shift-F7	Checks source file syntax. Same as Run->Syntax Check (see Run Menu, on page 498).
Shift-F8	Checks synthesis. Same as Run->Synthesis Check (see Run Menu, on page 498).
Shift-F10	Checks the timing constraints in the constraint files in your project and generates a report (<i>project_name_cck.rpt</i>). Same as Run->Constraint Check (see Run Menu, on page 498). 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 416).

Keyboard Shortcut	Description
Shift-F11	Toggles zooming out. Same as View->Zoom Out (see View Menu, on page 415).
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 551 .

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.



The following table describes the Project View buttons and options.

Button/Option	Action
Open Project...	Opens a new or existing project. Same as File->Open Project (see Open Project Command, on page 401).
Close Project	Closes the current project. Same as File->Close Project (see Run Menu, on page 498).
Add File...	Adds a source file to the project. Same as Project->Add Source File (see Build Project Command, on page 401).
Change File...	Replaces one source file with another. Same as Project ->Change File (see Change File Command, on page 429).
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 Implementation Options Command, on page 444).
Add P&R Implementation	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 626 for a description of the dialog box, and Running P&R Automatically after Synthesis, on page 1102 in the <i>User Guide</i> for information about using this feature.
View Log	Displays the log file. Same as View->View Log File (see View Menu, on page 415).
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.

Button/Option	Action
Auto Constrain	<p>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.</p> <p>See Using Auto Constraints, on page 492 in the <i>User Guide</i> for detailed information about using this option.</p> <p>You can also set this option on the Constraints panel of the Implementation Options dialog box.</p>
Automatic Compile Point	<p>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.</p> <p>See The Automatic Compile Point Flow, on page 627 in the <i>User Guide</i>.</p>
Continue on Error	<p><i>Synplify Premier</i></p> <p>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.</p> <p><i>Synplify Pro, Synplify Premier</i></p> <p>When enabled for compile-point synthesis, allows the operation to continue on error and synthesize the remaining compile points.</p>
FSM Compiler	<p>Turning on this option enables special FSM optimizations.</p> <p>Same as enabling the FSM Compiler option on the Options panel of the Implementation Options dialog box (see Optimizing State Machines, on page 584 in the <i>User Guide</i>).</p>
FSM Explorer	<p>When enabled, the FSM Explorer selects an encoding style for the finite state machines in your design.</p> <p>Same as enabling the FSM Explorer option on the Options panel of the Implementation Options dialog box. For more information, see Running the FSM Compiler, on page 585 in the <i>User Guide</i>.</p>

Button/Option	Action
Resource Sharing	<p>When enabled, makes the compiler use resource sharing techniques. This option does not affect resource sharing by the mapper.</p> <p>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 581 in the <i>User Guide</i> for usage details.</p>
Pipelining	<p>When enabled, uses register balancing and pipeline registers on multipliers and ROMs.</p> <p>Same as enabling the Pipelining option on the Options panel of the Implementation Options dialog box. See Pipelining, on page 559 in the <i>User Guide</i>.</p>
Retiming	<p>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.</p> <p>Same as enabling the Retiming option on the Options panel of the Implementation Options dialog box. Use the <code>syn_allow_retimings</code> attribute to enable or disable retiming for individual flip-flops. See syn_allow_retimings, on page 99 for syntax details.</p> <p>Note: Pipelining is automatically enabled when retiming is enabled.</p>
Run	<p>Runs synthesis (compilation and mapping).</p> <p>Same as the Run->Run command (see Run Menu, on page 498).</p>

CHAPTER 3

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 98
- [Schematic Objects and Their Display](#), on page 108
- [Basic Operations on Schematic Objects](#), on page 117
- [Multiple-sheet Schematics](#), on page 122
- [Exploring Design Hierarchy](#), on page 125
- [Filtering and Flattening Schematics](#), on page 133
- [Timing Information and Critical Paths](#), on page 139

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 98](#) and [Technology View, on page 100](#) 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:

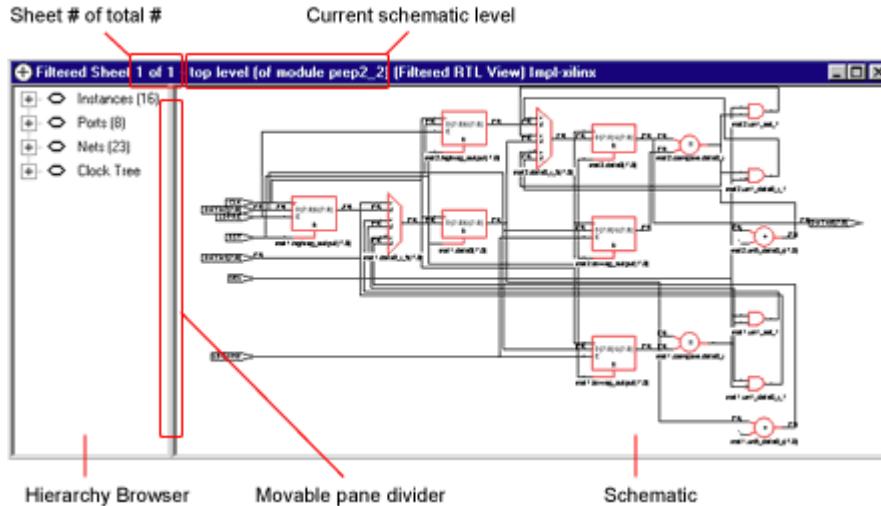
- [Hierarchy Browser, on page 102](#)
- [FSM Viewer Window, on page 103](#)
- [Filtered and Unfiltered Schematic Views, on page 105](#)
- [Accessing HDL Analyst Commands, on page 106](#)

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 100](#).

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 102](#). 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 102
Procedures for RTL view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Standard Schematic , on page 409 of the <i>User Guide</i> .
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool , on page 97
Commands for RTL view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands , on page 106 HDL Analyst Menu , on page 542

For information about ... See ...

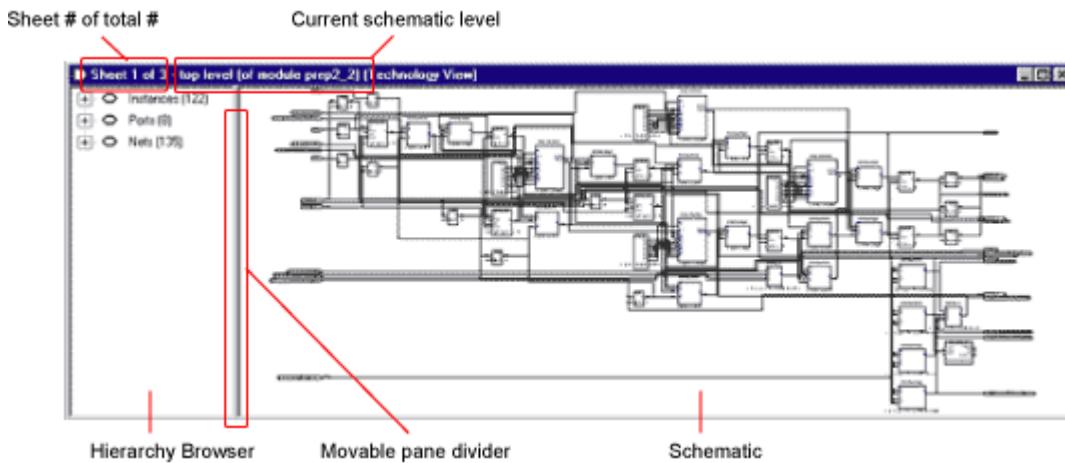
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands , on page 416
History commands: Back and Forward	View Menu: RTL and Technology Views Commands , on page 416
Search command	Find Command (HDL Analyst) , on page 407

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 98](#).

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 102](#). 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 102
Procedures for Technology view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Standard Schematic , on page 409 of the <i>User Guide</i>
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool , on page 97
Commands for Technology view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands , on page 106 HDL Analyst Menu , on page 542

For information about ... See ...

Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands , on page 416
History commands: Back and Forward	View Menu: RTL and Technology Views Commands , on page 416
Search command	Find Command (HDL Analyst) , on page 407

Hierarchy Browser

The Hierarchy Browser is the left pane in the RTL and Technology views. (See [RTL View](#), on page 98 and [Technology View](#), on page 100.) 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 103 for common symbols.

Instances	Lists all the instances and primitives in the design. In a Technology view, it includes all technology-specific primitives.
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 129 for more information about using the Hierarchy Browser for navigation and crossprobing.

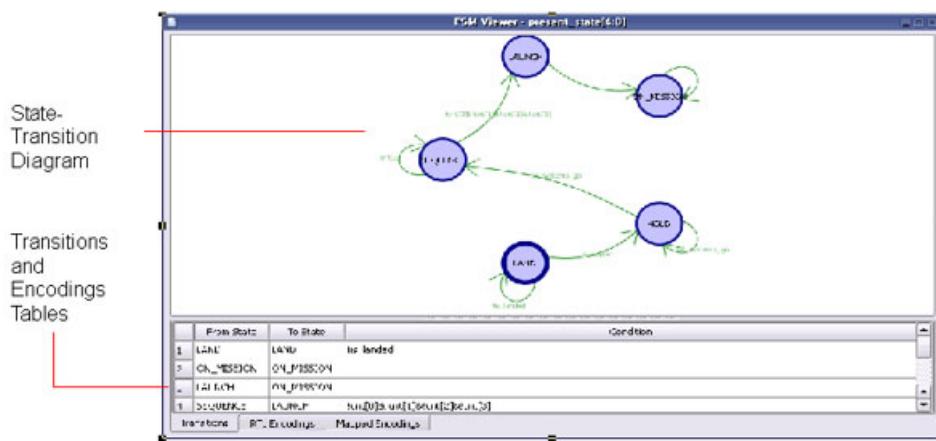
Hierarchy Browser Symbols

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

FSM Viewer Window

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.

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.

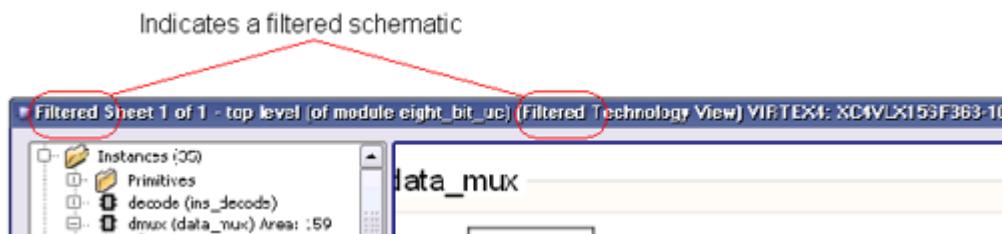
The following table describes FSM Viewer operations.

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.

Filtered and Unfiltered Schematic Views

HDL Analyst views ([RTL View, on page 98](#) and [Technology View, on page 100](#)) 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 lists 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. See the following topics:

- For a detailed description of filtering, see [Filtering and Flattening Schematics, on page 133](#).
- For procedures on using filtering, see [Filtering Schematics, on page 456](#) 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 542](#)
- [View Menu, on page 415](#)
- [RTL and Technology Views Popup Menus, on page 641](#)

- [Analyst Toolbar](#), on page 78

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 108
- [Sheet Connectors](#), on page 109
- [Primitive and Hierarchical Instances](#), on page 110
- [Hidden Hierarchical Instances](#), on page 113
- [Transparent and Opaque Display of Hierarchical Instances](#), on page 111
- [Schematic Display](#), on page 113

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

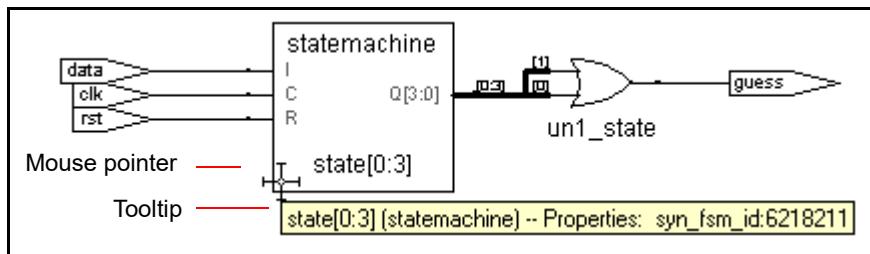
To obtain information about specific objects, you can view object properties with the **Properties** command from the right-click popup menu, or place the pointer over the object and view the object information displayed. With the latter method, information about the object displays in these two places 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:



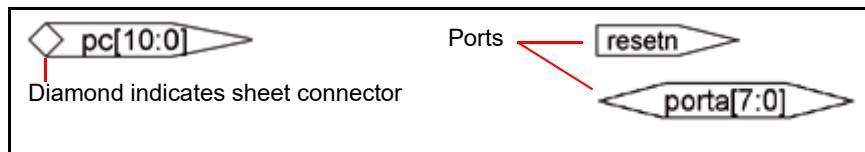
To disable tooltip display, select View -> Toolbars and disable the Show Tooltips option. Do this if you want to reduce clutter.

See also

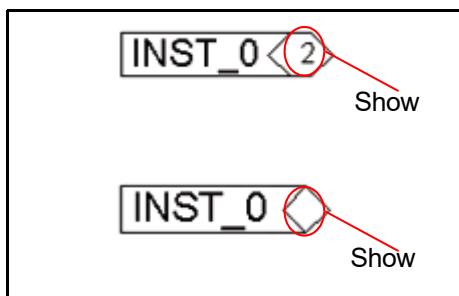
- [Pin and Pin Name Display for Opaque Objects](#), on page 114
- [Standard HDL Analyst Options Command](#), on page 589

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 594](#)) 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.



See also

- [Multiple-sheet Schematics](#), on page 122
- [Standard HDL Analyst Options Command](#), on page 589
- [RTL and Technology Views Popup Menus](#), on page 641

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	

In a schematic, logic gate primitives are represented with standard schematic symbols, and technology-specific primitives with various symbols (see [Hierarchy Browser, on page 102](#)). 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.

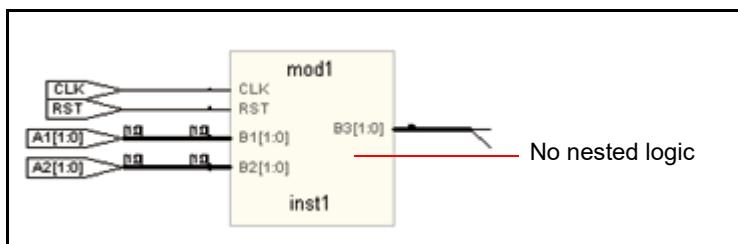
The Hierarchy Browser lists hierarchical instances under Instances, and uses this symbol:  . In a schematic, the display of hierarchical instances depends on the combination of the following:

- 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 111](#) 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 113](#) for more information.

Transparent and Opaque Display of Hierarchical Instances

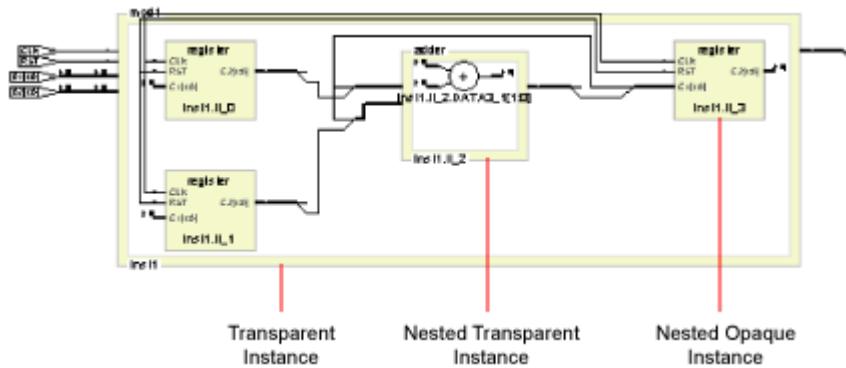
A hierarchical instance can be displayed transparently or opaquely. You cannot directly control the display; certain commands cause instances to be transparent. The distinction between transparent and opaque is important because some commands operate differently on transparent and opaque instances. For 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.



- 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 130](#) 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.



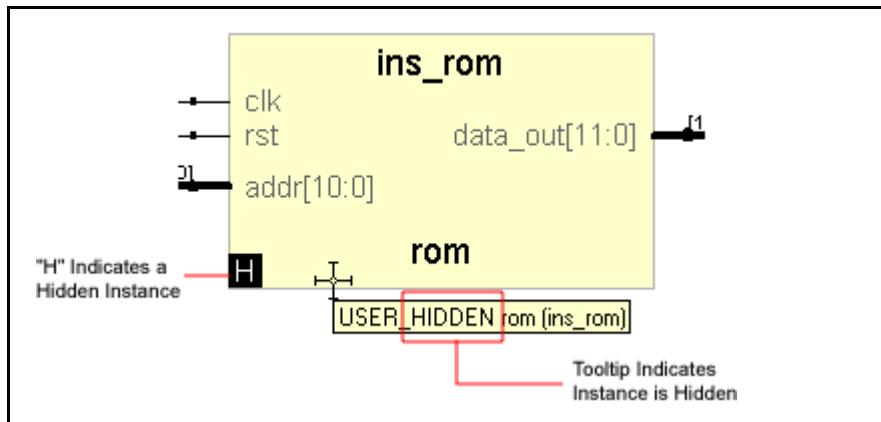
See also

- [Looking Inside Hierarchical Instances, on page 130](#)
- [Multiple Sheets for Transparent Instance Details, on page 124](#)
- [Filtered and Unfiltered Schematic Views, on page 105](#)

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 641](#)). 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

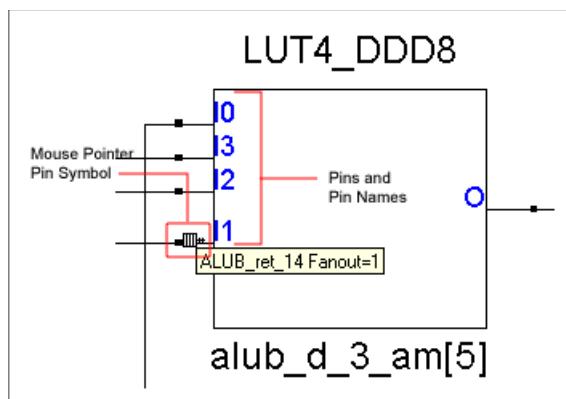
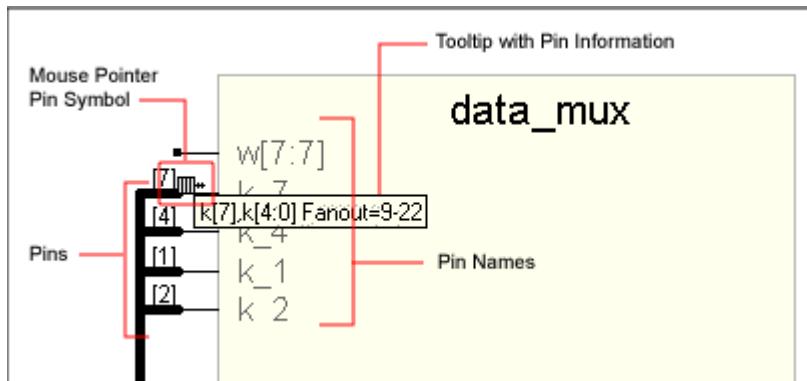
The HDL Analyst Options dialog box controls general properties for all HDL Analyst views, and can determine the display of schematic object information. Setting a display option affects all objects of the given type in all views. Some schematic options only take effect in schematic windows opened after 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 [Standard HDL Analyst Options Command, on page 589](#) for a complete list of display options.

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	Instances on a schematic by setting limits to the number of instances displayed
Filtered Instances	
Instances added for expansion	
Instance Name	Object labels
Show Conn Name	
Show Symbol Name	
Show Port Name	
Show Pin Name	Pin names. See Pin and Pin Name Display for Opaque Objects , on page 114 and Pin and Pin Name Display for Transparent Objects , on page 115 for details.
HDL Analyst->Show All Hier Pins	

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.



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.

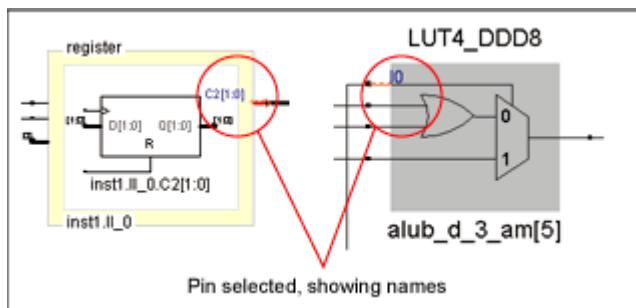
Transparent Hierarchical Instances

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.

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:

- [Standard HDL Analyst Options Command](#), on page 589
- [Controlling the Amount of Logic on a Sheet](#), on page 122
- [Analyzing Timing in Schematic Views](#), on page 474 in the *User Guide*

Basic Operations on Schematic Objects

Basic operations on schematic objects include the following:

- [Finding Schematic Objects](#), on page 117
- [Selecting and Unselecting Schematic Objects](#), on page 118
- [Crossprobing Objects](#), on page 119
- [Dragging and Dropping Objects](#), on page 121

For information about other operations on schematics and schematic objects, see the following:

- [Filtering and Flattening Schematics](#), on page 133
- [Timing Information and Critical Paths](#), on page 139
- [Multiple-sheet Schematics](#), on page 122
- [Exploring Design Hierarchy](#), on page 125

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 \(Standard\)](#), on page 431 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 431 in the *User Guide* for details.

- Edit -> Find command

The Edit -> Find command is described in [Find Command \(HDL Analyst\)](#), on page 407. 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.

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 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 433](#) 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 133](#) 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.

The following briefly list selection methods; for a concise table of selection procedures, see [Selecting Objects in the RTL/Technology Views, on page 417](#) 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 117](#)
- [HDL Analyst Menu, on page 542](#)

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

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 \(Standard\), on page 445](#) 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	<p><i>Synplify Pro, Synplify Premier</i> 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.</p> <p>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.</p> <p>To cross-probe from text other than source code, first select Options->HDL Analyst Options and then enable Enhanced Text Crossprobing.</p>
FSM Viewer	Analyst view	<p>The target view must be open. The state machine must be encoded with the onehot style to crossprobe from the transition table.</p> <p>Select a state anywhere in the FSM Viewer (bubble diagram or transition table). The corresponding object is automatically selected in the HDL Analyst view.</p>

From ...	To ...	Do this ...
Analyst view	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.
FSM Viewer		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. <i>Synplify Pro, Synplify Premier</i> If the target view is the FSM Viewer, 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

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 109](#)) mark logic connections from one sheet to the next.

For more information, see

- [Controlling the Amount of Logic on a Sheet, on page 122](#)
- [Navigating Among Schematic Sheets, on page 122](#)
- [Multiple Sheets for Transparent Instance Details, on page 124](#)

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:

- [Standard HDL Analyst Options Command, on page 589](#)
- [Setting Schematic Preferences, on page 420 of the User Guide.](#)

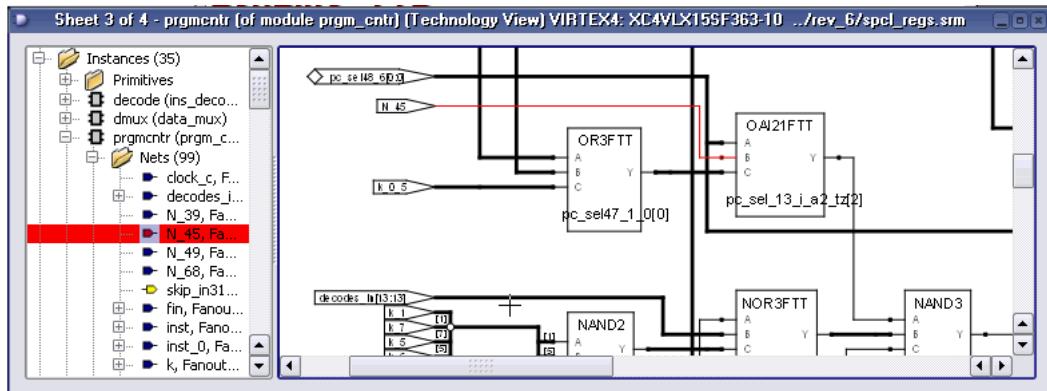
Navigating Among Schematic Sheets

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.

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 418](#) 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.

See also:

- [Filtering and Flattening Schematics, on page 133](#)

- [View Menu: RTL and Technology Views Commands, on page 416](#)
- [Pushing and Popping Hierarchical Levels, on page 125](#)

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 122](#).
- 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 122](#)
- [View Menu: RTL and Technology Views Commands, on page 416](#)

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 125
- [Navigating With a Hierarchy Browser](#), on page 129
- [Looking Inside Hierarchical Instances](#), on page 130

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 129 and [Looking Inside Hierarchical Instances](#), on page 130 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.

When you push/pop, the HDL Analyst window displays the appropriate level of design hierarchy, except in the following cases:

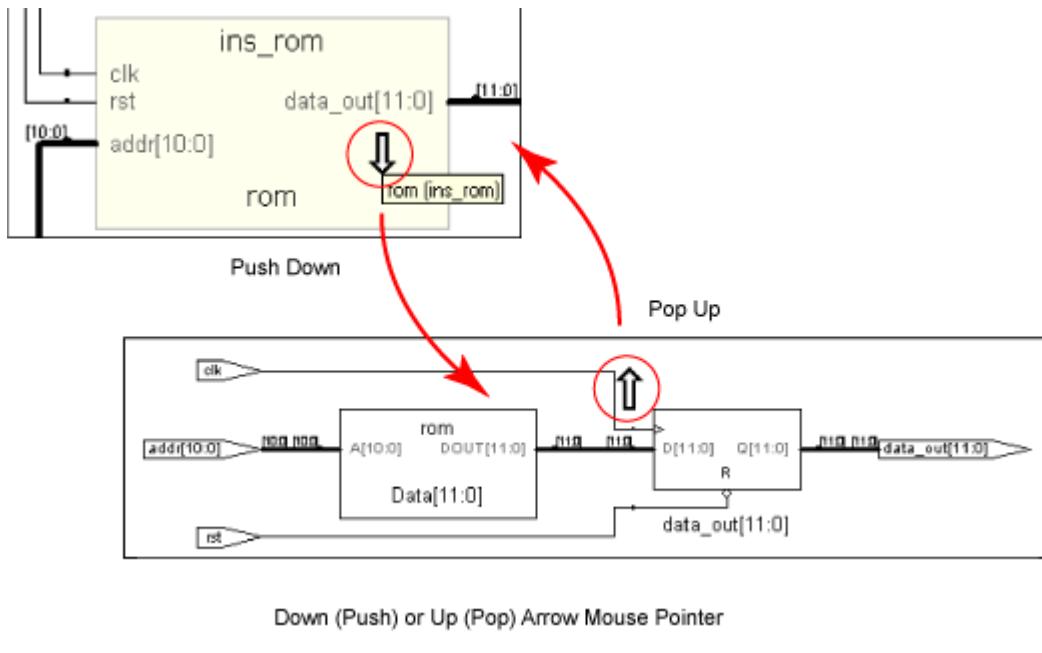
- When you push into an inferred state machine in an RTL view, in the Synplify tool, the `statemachine.info` text file opens. In the Synplify Pro and Synplify Premier tools, the FSM Viewer opens, with graphical information about the FSM. See the [FSM Viewer Window, on page 103](#), for more information.
- 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 126](#) for details.
- A small H symbol () 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.

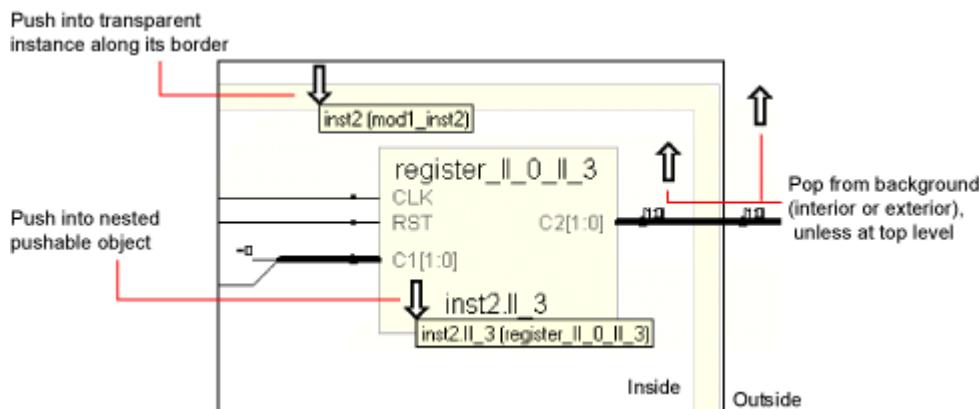
How to Push and Pop Hierarchical Levels

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.



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.ll_3` descends two levels.



The following arrow mouse pointers indicate status in Push/Pop mode. For other indicators, see [Pushable Schematic Objects](#), on page 125.

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:

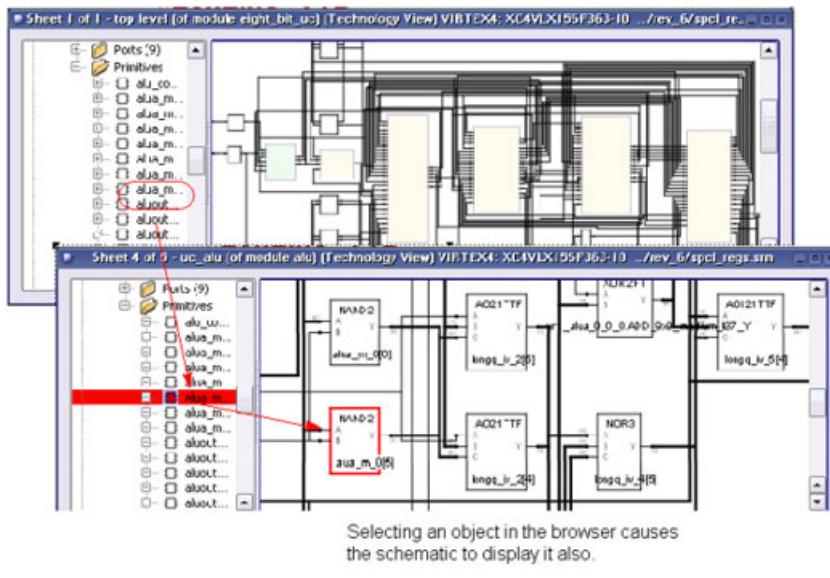
- [Hidden Hierarchical Instances](#), on page 113
- [Transparent and Opaque Display of Hierarchical Instances](#), on page 111
- [Using Mouse Strokes](#), on page 72
- [Navigating With a Hierarchy Browser](#), on page 129

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 125](#) and [Looking Inside Hierarchical Instances, on page 130](#) 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.



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.

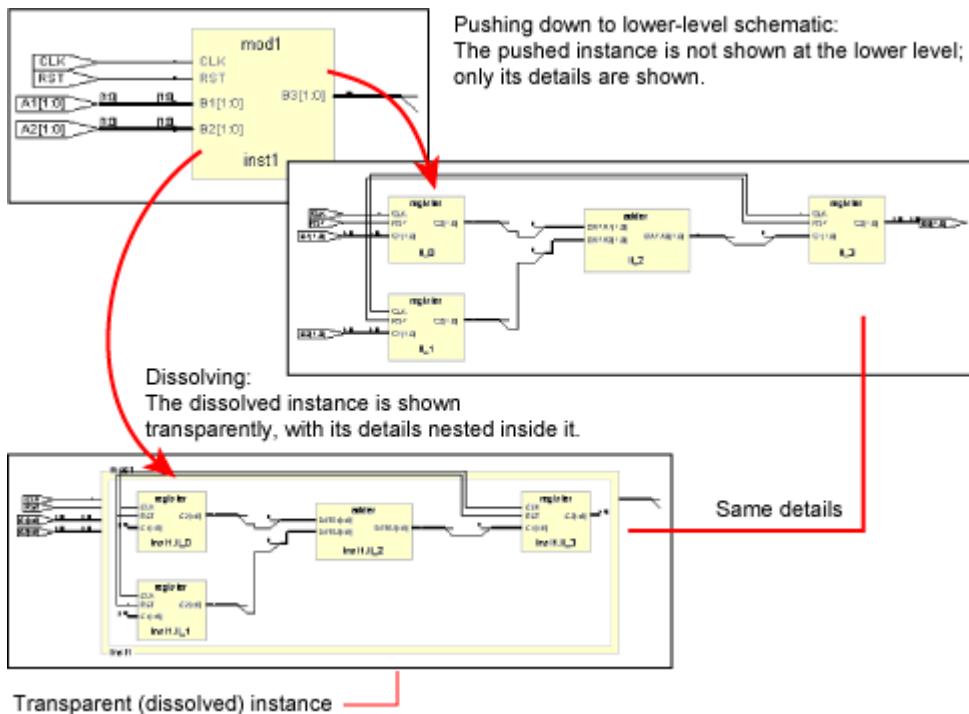
See also:

- [Crossprobing Objects](#), on page 119
- [Pushing and Popping Hierarchical Levels](#), on page 125
- [Hierarchy Browser Popup Menu Commands](#), on page 641

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 129 and [Navigating With a Hierarchy Browser](#), on page 129 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 133](#) 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 125](#)
- [Navigating With a Hierarchy Browser, on page 129](#)
- [HDL Analyst Command, on page 543](#)
- [Transparent and Opaque Display of Hierarchical Instances, on page 111](#)
- [Hidden Hierarchical Instances, on page 113](#)

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 133
- [Combined Filtering Operations](#), on page 134
- [Returning to The Unfiltered Schematic](#), on page 134
- [Commands That Flatten Schematics](#), on page 135
- [Selective Flattening](#), on page 136
- [Filtering Compared to Flattening](#), on page 137

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).

For information about filtering procedures, see [Filtering Schematics, on page 456](#) in the *User Guide*.

See also:

- [Filtered and Unfiltered Schematic Views](#), on page 105
- [HDL Analyst Menu](#), on page 542 and [RTL and Technology Views Popup Menus](#), on page 641

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 546](#) 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 544](#) 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 544](#) 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 549](#) 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 117](#) for more information.

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 641
- [View Menu: RTL and Technology Views Commands](#), on page 416

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.	

Command	Unfiltered Schematic	Filtered Schematic
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 136](#)).

See also:

- [Filtering Compared to Flattening, on page 137](#)
- [Selective Flattening, on page 136](#)

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 641](#) 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 <i>everything</i> at the current level and below	Flattens only the non-hidden, <i>transparent</i> 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.

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 641
- [Selective Flattening](#), on page 136

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 139
- [Critical Paths and the Slack Margin Parameter](#), on page 140
- [Examining Critical Path Schematics](#), on page 141

See the following for more information about timing and result analysis:

- [Watch Window](#), on page 54
- [Log File](#), on page 199
- [Chapter 14, 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 453](#)). 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 519
- [Timing Reports](#), on page 205
- [Log File](#), on page 199

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 resynthesizes 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.

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 542
- [HDL Analyst Command](#), on page 543
- [Handling Negative Slack](#), on page 480 of the *User Guide*
- [Analyzing Timing in Schematic Views](#), on page 474 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.

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 134
- [Filtering and Flattening Schematics](#), on page 133

CHAPTER 4

Constraint Guidelines

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 144
- [Constraint Files](#), on page 145
- [Timing Constraints](#), on page 147
- [FDC Constraints](#), on page 150
- [Methods for Creating Constraints](#), on page 151
- [Constraint Translation](#), on page 153
- [Constraint Checking](#), on page 176
- [Database Object Search](#), on page 178
- [Forward Annotation](#), on page 179
- [Auto Constraints](#), on page 179

Constraint Types

One way to ensure the FPGA synthesis tool achieves 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 (<code>create_clock</code>), clock groups (<code>set_clock_groups</code>), and timing exceptions like multicycle and false paths (<code>set_multicycle_path...</code>). See Timing Constraints , on page 147 for information on defining these constraints.
Design	Additional design goals that enhance or guide tool optimizations. Examples: Attributes and directives (<code>define_attribute</code> , <code>define_global_attribute</code>), I/O standards (<code>define_io_standard</code>), and compile points (<code>define_compile_point</code>).
Physical	Constraints that provide placement optimizations for synthesis. Examples: Region constraints for floorplanning (<code>create_region</code>).

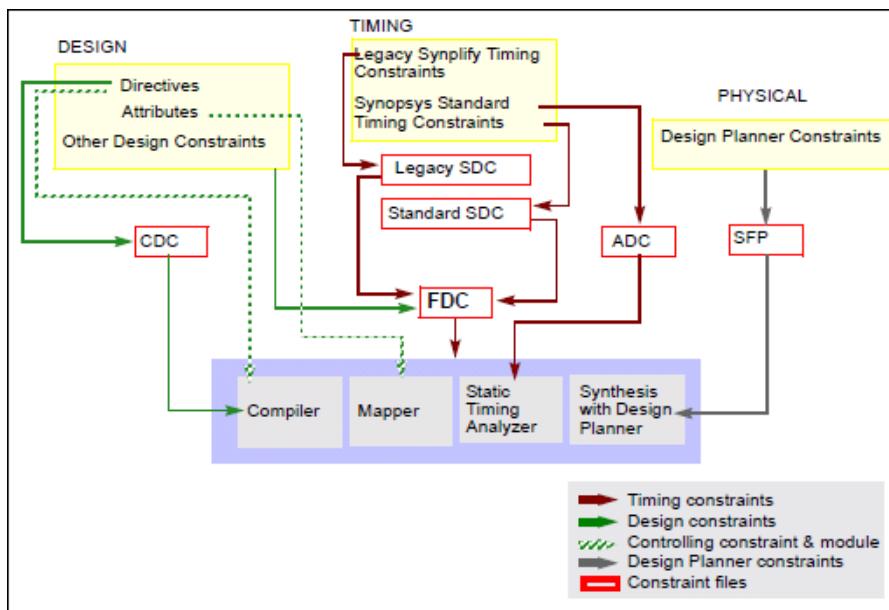
The easiest way to specify constraints is through the SCOPE interface. The tool saves timing and design constraints to an FDC file that you add to your project.

See Also

Constraint Files , on page 145	Overview of constraint files
Timing Constraints , on page 147	Overview of timing constraint definitions and FDC file generation.
SCOPE Constraints Editor, on page 294	Information about automatic generation of timing and design constraints.
Timing Constraints, on page 342	Timing constraint syntax
Design Constraints, on page 391	Design constraint syntax

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 147](#). The figure also indicates the specific processes controlled by attributes and directives.



The table is a summary of the various kinds of constraint files.

File	Type	Common Commands	Comments
FDC	Timing constraints	create_clock, set_multicycle_delay ...	Used for synthesis. Includes timing constraints that follow the Synopsys standard format as well as design constraints.
	Design constraints	define_attribute, define_io_standard ...	
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.
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.

Timing Constraints

The synthesis tool has 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 147
- [FDC File Generation](#), on page 148
- [Timing Constraint Precedence in Mixed Constraint Designs](#), on page 148

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.

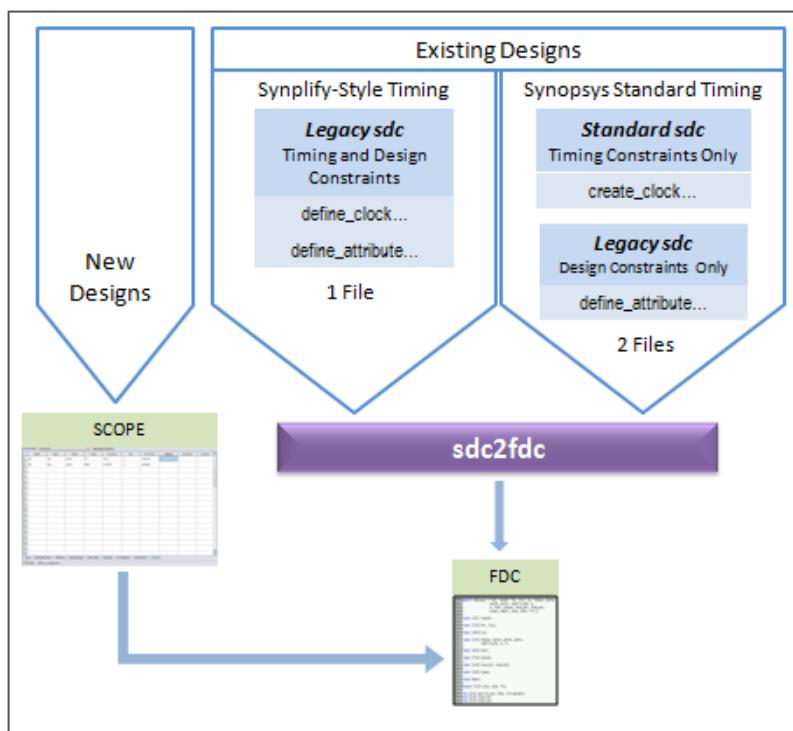
Starting with the G-2012.09 release, Synopsys standard timing constraint format has replaced the legacy-style constraint format, and a new FDC (FPGA design constraint) file consolidates both timing and design formats. As a result of these updates, there are some changes in the use model:

- 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_multicycle_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 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.

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. This means that constraint priority is determined by the constraint that is read last.

With the legacy timing constraints, it is strongly recommended that you convert them to the `fdc` format. However, 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 [Timing Constraints, on page 342](#).

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 151](#) 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 247](#) for details on how to run `sdc2fdc`.

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	<p>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 4, Constraint Commands, for information how to use SCOPE to automatically generate constraint syntax.</p> <p>Access: File->New->FPGA Design Constraints ...</p>
Manually-Entered Text Editor (fdc File, all other constraint files)	<p>You can manually enter constraints in a text file. Make sure to use the correct syntax for the timing and design commands.</p> <p>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 319.</p> <p>You can also open any constraint file in a text editor to modify it.</p>
Source Code Attributes/Directives (HDL files, cdc file)	<p>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.</p> <p>You can specify some compiler directives in a cdc file with the New-> Compiler Directives command.</p> <p>Attributes can be entered through the SCOPE interface, as they affect the mapper, not the compiler</p>

Definition Method	Description
Automatic—First Pass	<p>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 492 in the <i>User Guide</i> for details.</p> <p>Use this method as a quick first pass to get an idea of what constraints can be set.</p>

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 338](#) to determine the constraint that takes precedence.

See Also

To specify the correct syntax for the timing and design commands, see:

- [Chapter 4, Constraint Commands](#)
- [Attribute Reference Manual](#)

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 247](#)) 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 [Using the SCOPE Editor \(Legacy\), on page 249](#).

See Also

To use the current SCOPE editor, see:

- [Chapter 4, Constraint Commands](#)
- [Chapter 5, Specifying Constraints](#)

Constraint Translation

Intel, Xilinx

The tool includes standalone scripts to convert specific vendor constraints, as well as functionality that includes constraint translation as part of the larger task of generating a synthesis project from vendor files.

For older vendor constraints, translation is a two-step process. You must first use the appropriate utility or command to translate the constraints to legacy sdc, and then migrate the constraints to the fdc constraint format with the sdc2fdc script.

Conversion	For details, see
Legacy SDC to FDC	sdc2fdc Conversion, on page 154 Converting SDC to FDC, on page 247 in the <i>User Guide</i>
Intel	
QSF (file)	qsf2sdc Conversion, on page 160 Translating Intel FPGA QSF Constraints, on page 267 in the <i>User Guide</i>
QSF (project)	qsf2syn, on page 125 in the <i>Command Reference</i> Import Intel QSF Project Command, on page 495 in the <i>Command Reference</i> Importing Projects from Quartus, on page 717 in the <i>User Guide</i>
SOPC project	sopc2syn Conversion, on page 158 Import Intel SOPC Project Command, on page 492 in the <i>Command Reference</i> Working with SOPC Builder Components, on page 713 in the <i>User Guide</i>

Xilinx

EDK/ISE project	ise2syn , on page 95 in the <i>Command Reference</i> Import Xilinx EDK/ISE Project Command , on page 489 in the <i>Command Reference</i> Converting Xilinx Projects with ise2syn , on page 761 in the <i>User Guide</i>
UCF constraints	UCF Conversion , on page 162 Converting and Using Xilinx UCF Constraints , on page 275 in the <i>User Guide</i>
XDC project	add_vivado_ip , on page 31 in the <i>Command Reference</i> Add Vivado IP Command , on page 486 in the <i>Command Reference</i> Incorporating Vivado IP , on page 740 in the <i>User Guide</i>

sdc2fdc Conversion

The `sdc2fdc` Tcl shell command translates legacy FPGA timing constraints to Synopsys FPGA timing constraints. This command scans the input SDC files and attempts to convert constraints for the implementation.

For details, see the following:

- [Troubleshooting Conversion Error Messages](#), on page 154
- [sdc2fdc FPGA Design Constraint \(FDC\) File](#), on page 156
- [sdc2fdc](#), on page 147 in the *Command Reference* manual

Troubleshooting Conversion Error Messages

The following table contains common error messages you might encounter when running the `sdc2fdc` Tcl shell command, and descriptions of how to resolve these problems. In addition to these messages, you must also ensure that your files have read/write permissions set properly and that there is sufficient disk space.

Message Example	Underlying Problem
Remove/disable D:FDC_constraints/rev_FDC/top_translated.fdc from the current implementation.	Cannot translate a *_translated.fdc file

Message Example**Underlying Problem**

Add/enable one or more SDC constraint files.

No active constraint files

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

Clock not translated

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

Clock not translated

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 -from list for
 define_multicycle_path {a* b*}

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 -to list for
 define_multicycle_path {i: *y* .q[*] p:ena}

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

Bad -from list for
 define_multicycle_path
 {i:/*y* .q[*] p:ena enab}

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

No period or frequency found

`"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

Must specify both -rise and
 -fall, or neither

Fix any issues in the SDC source file and rerun the sdc2fdc command.

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 to this effect displays in the Tcl shell window.

Discontinued Support for Legacy FPGA Constraints (Intel)

The legacy SDC format is not supported for Agilex, Stratix 10, Arria 10, Stratix V, and Cyclone 10-GX devices in the new Intel mapper. These devices support the FDC format constraints only. To migrate from SDC to FDC, run the `sdc2fdc` utility from the Tcl window. See *Command Reference Guide->Specifying Constraints->Converting SDC to FDC*.

sdc2fdc FPGA Design Constraint (FDC) File

The FDC constraint file generated after running `sdc2fdc` contains translated legacy FPGA timing constraints (SDC), which are now in the FDC format. This file is divided into two sections:

- 1 Contains this information:
 - Valid FPGA design constraints (e.g. `define_scope_collection` and `define_attribute`)
 - Legacy timing constraints that were not translated because they were specified with `-disable`.
 - 2 Contains the legacy timing constraints that were translated.
-

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
#####
#####
#####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 clkbr:r
define_input_delay -disable {d[7:0]} 0.30 -ref clkbr: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 {LVCMS_33}#
#
#I/O standards
#
#
##Compile Points
#
#
##Other Constraints
#####
#SDC compliant constraints translated from Legacy Timing Constraints
#####
#
set_rtl_ff_names {#}

create_clock -name {clk_a} [get_ports {clk_a}] -period 10 -waveform {0 5.0}
create_clock -name {clk_b} [get_ports {clk_b}] -period 6.666666666666667
    -waveform {0 3.333333333333335}
set_input_delay -clock [get_clocks {clk_a}] -clock_fall -add_delay 0.000 [all_inputs]
set_output_delay -clock [get_clocks {clk_a}] -add_delay 0.000 [all_outputs]
set_input_delay -clock [get_clocks {clk_a}] -add_delay 2.00 [get_ports {a[7:0]}]
set_input_delay -clock [get_clocks {clk_a}] -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 {clk_a dcm|clk0_derived_clock dcm| \
        clk2x_derived_clock dcm|clk0fx_derived_clock}]
set_clock_groups -name default_clkgroup_1 -asynchronous \
    -group [get_clocks {clk_b}]


```

sopc2syn Conversion

Intel FPGA

The **sopc2syn** utility is the batch mode equivalent of the Import IP->Import Intel FPGA SOPC Project command and lets you include subsystems created with the Intel 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. It also 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.

This section describes salient features of the `sopc2syn` utility:

Input files	<p>The utility reads the following files to extract required information and construct a synthesis project:</p> <ul style="list-style-type: none"> • <code>ptf</code> file • <code>scr</code> file (script file) • <code>qsf</code>, <code>edf</code>, and <code>bsf</code> files from the SOPC Builder project
Output files	<p>The utility generates the following output files:</p> <ul style="list-style-type: none"> • Synplify project file (<code>prj</code>) • Synplify Constraint file (<code>sdc</code>) file • Core wrapper file for black boxes • Directories such as the unused directory containing the HDL files used in the project, and the <code>quartus_synthesis</code> directory which contains the HDL files for the encrypted cores and forced black boxes. • PAR directory files: <code>qsf</code>, <code>qof</code>, <code>bdf</code>, <code>bsf</code>
Initialization files	<p>The initialization file for <code>sopc2syn</code> is called <code>sopc2syn.ini</code>. This file can be located in two places:</p> <ul style="list-style-type: none"> • <code>synplifyInstallDirectory/lib/sopc2syn.ini</code> • <code>SOPC_Builder_proj_dir/sopc2syn.ini</code> <p>The <code>ini</code> 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 <code>ini</code> files.</p> <p>If you have two <code>ini</code> files, the file in <code>SOPC_Builder_proj_dir</code> overrides the one in <code>synplifyInstallDirectory</code>. Use this fact to manage project settings. Put general project settings for all Synplify projects generated from <code>sopc2syn.exe</code> in <code>synplifyInstallDirectory/lib/sopc2syn.ini</code>. Put project settings for specific projects in the project directory <code>ini</code> file.</p>
<code>sopc2syn</code> syntax	See sopc2syn , on page 197 of the <i>Command Reference</i> manual.

qsf2sdc Conversion

The `qsf2sdc` utility converts constraints from a Quartus settings file (`qsf`) into a Synopsys FPGA `sdc` constraints file. Run the utility from any shell window, using the syntax described in [qsf2sdc, on page 124](#) in the *Command Reference*.

Currently, this script only translates pin location and I/O standard constraints from the QSF file. The following sections describe the details:

- [QSF Pin Location Constraints, on page 160](#)
- [QSF I/O Standard Constraints, on page 160](#)
- [Example of Translated QSF Constraints, on page 161](#)

QSF 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 after running `qsf2sdc`. See [Example of Translated QSF Constraints, on page 161](#) for a file example.

QSF Syntax	SDC Syntax
<code>set_location_assignment -to CLK PIN_P23</code>	<code>define_attribute {CLK} syn_loc {PIN_P23}</code>
<code>set_location_assignment PIN_P23 -to CLK</code>	<code>define_attribute {CLK} syn_loc {PIN_P23}</code>
<code>set_location_assignment PIN_AA11 -to TXWRADDR[10]</code>	<code>define_attribute {TXWRADDR[10]} syn_loc {PIN_AA11}</code>
<code>set_location_assignment -to TXWRADDR[10] PIN_AA11</code>	<code>define_attribute {TXWRADDR[10]} syn_loc {PIN_AA11}</code>

QSF 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 after running `qsf2sdc`. See [Example of Translated QSF Constraints, on page 161](#) for a file example.

QSF Syntax	SDC Syntax
<code>set_instance_assignment -name IO_STANDARD -to CLK LVTTL</code>	<code>define_attribute {CLK} syn_pad_type {LVTTL}</code>
<code>set_instance_assignment -name IO_STANDARD LVTTL -to CLK</code>	<code>define_attribute {CLK} syn_pad_type {LVTTL}</code>
<code>set_instance_assignment -name IO_STANDARD -to TXWRADDR[10] LVTTL</code>	<code>define_attribute {TXWRADDR[10]} syn_pad_type {LVTTL}</code>
<code>set_instance_assignment -name IO_STANDARD LVTTL -to TXWRADDR[10]</code>	<code>define_attribute {TXWRADDR[10]} syn_pad_type {LVTTL}</code>

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
```

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
```

UCF Conversion

Xilinx

Some parts of the UCF conversion, like the translation of constraints, is used by other utilities, like ise2syn.

For more information about this functionality, see the following:

- [UCF Supported Constraints Summary](#), on page 162
- [Examples of Translated UCF Constraints](#), on page 164
- [Unsupported UCF Constraints](#), on page 174
- [Converting and Using Xilinx UCF Constraints](#), on page 275 in the *User Guide*

UCF Supported Constraints Summary

This section summarizes of the UCF constraints translated to SDC. Refer to the Xilinx documentation for syntax details.

Constraint	Description
Grouping Constraints	<p>The following constraints specify how objects are grouped:</p> <ul style="list-style-type: none"> • 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 165 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 166 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 166 for an example.
LOC	<p>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 167 for examples of single-location, multiple-location, and range-of-locations LOC constraints.</p>
MAX DELAY	<p>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 169 for examples.</p>

Constraint Description

OFFSET	<p>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.</p> <p>This constraint allows you to</p> <ul style="list-style-type: none"> • 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. <p>See OFFSET Constraint Translation Examples , on page 170 for examples, and Unsupported UCF Constraints , on page 174 for information about currently unsupported constraints.</p>
PERIOD	<p>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.</p> <p>The PERIOD constraint is applied to the clock net in the UCF file. See PERIOD Constraint Translation Examples , on page 170 for examples.</p>
PROP	<p>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 172 for examples of general, instantiated RAM, IOB, and DCM PROP constraints.</p>
TIG	<p>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 173 for examples.</p>

Examples of Translated UCF Constraints

See the following for examples of translated constraints:

- [TNM Constraint Translation Example , on page 165](#)
- [TNM_NET Constraint Translation Example , on page 166](#)
- [TIMEGRP Constraint Translation Example , on page 166](#)
- [LOC Constraint Translation Examples , on page 167](#)

- [MAXDELAY Constraint Translation Examples](#), on page 169
- [OFFSET Constraint Translation Examples](#), on page 170
- [PERIOD Constraint Translation Examples](#), on page 170
- [PROP Constraint Translation Examples](#), on page 172
- [TIG Constraint Examples](#), on page 173

TNM Constraint Translation Example

UCF Syntax	Converted SDC Constraint
NET "clk1" TNM = "mytn1";	define_scope_collection mytn1_0_0 {find -net clk1}
NET "clk2" TNM = "mytn1";	define_scope_collection mytn1_0_fr {expand -hier
TIMESPEC "TS_mytn1" =	-from \$mytn1_0_0}
PERIOD "mytn1" 5 ns;	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 166 for the syntax for predefined groups.

TNM_NET Constraint Translation Example

UCF Syntax	Converted SDC Constraint
<pre>NET "clk1" TNM_NET = FFS (*2) "mytn1"; TIMESPEC "TS_mytn1" = TO "mytn1" 5 ns;</pre>	<pre>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_0_ff_plus {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 166 for the syntax for predefined groups.</pre>

TIMEGRP Constraint Translation Example

UCF Syntax	Converted SDC Constraint
<pre>TIMEGRP "group1" = FFS; TIMESPEC "TS_group1" = PERIOD "group1" 5 ns;</pre>	<pre>define_scope_collection group1 {c_union \$FFS \$FFS} define_path_delay -to {{\$group1}} -max 5.000 See Predefined Groups , on page 166 for the syntax for the predefined group \$FFS.</pre>

Predefined Groups

The following table describes the FDC syntax for predefined groups.

CPUS

```
define_scope_collection CPUS {find -hier -inst * -filter @view==PPC405 ||
@view==PPC405_*}
```

DSPS

```
define_scope_collection DSPS {find -hier -inst * -filter @view==DSP48}
```

FFS

```
define_scope_collection FFS {find -hier -inst * -filter @view==dff* || @view==sdff* || @view==FD*}
```

HSIOS

```
define_scope_collection HSIOS {find -hier -inst * -filter @view==GT11 || @view==GT11_* || @view==GT10 || @view==GT10_*}
```

LATCHES

```
define_scope_collection LATCHES {find -hier -inst * -filter @view==lat || @view==latr || @view==lats || @view==LD*}
```

MULTS

```
define_scope_collection MULTS {find -hier -inst * -filter @view==MULT18X18S}
```

PADS

```
define_scope_collection PADS {find -port *}
```

RAMS

```
define_scope_collection RAMS {find -hier -inst * -filter @view==ram1 || @view==nram || @view==RAM*}
```

LOC Constraint Translation Examples

The following examples show how LOC constraints for single and multiple locations are translated:

- Single Location (Net)

The following shows how a LOC constraint for a single location attached to a net is translated:

UCF Syntax

NET "clock" LOC = "C2";

Converted SDC Constraint

```
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}

- 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

```
NET "dout1" TPTHRU = "thru1";
TIMESPEC "TS_basic1" = FROM FFS
THRU "thru1" TO PADS 4.5;
```

```
TIMESPEC "TS_group1" = FROM RAMS
5;
```

Converted SDC Constraint

```
define_scope_collection thru1 {find -net dout1}
define_path_delay -from {{$FFS}} -through
{$thru1} -to {{$PADS}} -max 4.500
```

```
define_path_delay -from {{$RAMS}} -max
5.000
```

The following shows how TNM-based MAXDELAY TPTHRU constraints are translated:

UCF Syntax

```
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;
```

Converted SDC Constraint

```
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
```

UCF Syntax

```
NET "din" TNM = "din_ff";
PIN "out.D" TPTHRU = "mypi";
TIMESPEC "TS_basic2" =
MAXDELAY
FROM "din_ff" THRU "mypi"
333333 KHz;
```

Converted SDC Constraint

```
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
$din_ff_0_fr}
define_scope_collection mypi {find -pin out.D}
define_path_delay -from {$din_ff} -through {$mypi}
-max 3.000
#WARNING# PIN <pin> TPTHRU is supported only
for pins of instantiated components.
```

OFFSET Constraint Translation Examples

The following examples show how various OFFSET constraints are translated.

- Global OFFSET

UCF Syntax

```
OFFSET = OUT 40 ns AFTER "clk";
```

Converted SDC Constraint

```
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

```
NET "sften" OFFSET = IN : 6ns
:
BEFORE : clk;
```

Converted SDC Constraint

```
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).

PERIOD Constraint Translation Examples

The following examples show how various PERIOD constraints are translated.

- PERIOD in TIMESPEC

UCF Syntax	Converted SDC Constraint
<pre>NET:"clock":TNM_NET = "clock"; TIMESPEC:"TS_clock" = PERIOD:"clock":50:ns:HIGH:40%: INPUT_JITTER:50:ps:PRIORITY:5;</pre>	<pre>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%;</pre> <p>Synthesis ignores INPUT_JITTER and PRIORITY, which is why the nearest equivalents are used.</p>
<pre>TIMEGRP "group1" = RAMS; TIMESPEC "TS_group1" = PERIOD "group1"5.0 ns HIGH 50.00%;</pre>	<pre>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</pre>

- PERIOD Attached to a Net

UCF Syntax	Converted SDC Constraint
<pre>NET "clock":PERIOD = 28 ns: HIGH:60%INPUT_JITTER:10:ps ; ;</pre>	<pre>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%;</pre> <p>Synthesis ignores INPUT_JITTER, which is why the nearest equivalents are used.</p>

- Derived Clocks

UCF Syntax	Converted SDC Constraint
<pre>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;</pre>	<pre>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;</pre> <p>Synthesis ignores INPUT_JITTER and PRIORITY, which is why the nearest equivalents are used.</p>

PROP Constraint Translation Examples

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}

This is an example of a RAM PROP constraint:

UCF Syntax	Converted SDC Constraint
INST "instRAM" WRITE_MODE_A = READ_FIRST;	define_attribute {i:instRAM} WRITE_MODE_A {READ_FIRST}

This is an example of an IOB PROP constraint:

UCF Syntax	Converted SDC Constraint
NET "in" IOSTANDARD = AGP ;	define_io_standard {in} syn_pad_type {AGP}

This is an example of a DCM PROP constraint:

UCF Syntax	Converted SDC Constraint
INST "dcm1" CLK_FEEDBACK = 1X	define_attribute {dcm1} CLK_FEEDBACK {1X}

TIG Constraint Examples

This is a global TIG example:

UCF Syntax	Converted SDC Constraint
NET "reset_l" TIG;	define_false_path -through {n:reset_l}

This is a single_TSID TIG example:

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.

This is a multiple_TSID TIG example:

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.

Unsupported UCF Constraints

The following types of OFFSET constraints are currently not translated:

- [OFFSET Using Keyword VALID](#), on page 174
- [OFFSET Using HIGH/LOW Clock Edge](#), on page 174

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

```
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 ;
```

Converted SDC Constraint

#WARNING# Keyword HIGH/LOW defining "CLK edge" is not supported.

Constraint Checking

The synthesis tool has 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 standalone timing analyzer (STA) generates a customized timing report when you need more details about specific paths 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

Note: Using collections with Tcl control constructs (such as if, for, foreach, and while) can produce unexpected synthesis results. Avoid defining constraints for collections with control constructs, especially since the constraint checker does not recognize these built-in Tcl commands.

See [Constraint Checking Report, on page 218](#) for details.

Timing Constraint Report Files

The results of running constraint checking, synthesis, and standalone 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 218).
_cck_fdc_rpt	Lists the wildcard expansion results of running the constraint checker for collections with the get_* and all_* object query commands using the check_fdc_query Tcl command. See check_fdc_query, on page 40 for more information.
scck.rpt	Lists the results of running the constraint checker for collections with the get* and all_* object query commands.
.ta	Reports timing analysis results (see Generating Custom Timing Reports with STA, on page 482).
.srr or .htm	Reports post-synthesis timing results as part of the text or HTML log file (see Timing Reports, on page 205 and Log File, on page 199).

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 tool provides 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 236 and find, on page 221 .
Collections	define_collection, c_union...	Create, copy, evaluate, traverse, and filter collections. See Using Collections, on page 236 and Collection Commands, on page 240 for more information.

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
Intel	ACF SCF TCL
Lattice	LPF
Microchip PolarFile	_VM.SDC
Microchip SmartFusion2	_SDC.SDC _VM.SDC
Microchip All devices except PolarFile and SmartFusion2	_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. For details, see [Using Auto Constraints, on page 492](#) in the *User Guide*.

CHAPTER 5

Input and Result Files

This chapter describes the input and output files used by the tool.

- [Input Files](#), on page 182
- [Libraries](#), on page 187
- [Output Files](#), on page 191
- [Log File](#), on page 199
- [Timing Reports](#), on page 205
- [Hierarchical Area Report](#), on page 215
- [Constraint Checking Report](#), on page 218

Input Files

The following table describes the input files used by the synthesis tool.

Extension	File	Description
.adc	Analysis Design Constraint	<p>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.</p> <p>You can forward annotate adc constraints to your vendor constraint file without rerunning synthesis. See Using Analysis Design Constraints, on page 485 of the <i>User Guide</i> for details.</p>
.cdc	Compiler Design Constraint	<p><i>Synplify Premier</i></p> <p>Specifies directives to be applied to the source code. See Specifying Directives in a CDC File, on page 141 of the <i>User Guide</i> for information about specifying the directives and using this file. See The Compiler Directives File, on page 12 for examples.</p>
.dpf	Design Plan	<p><i>Synplify Premier with Design Planner</i> Intel Stratix V; Xilinx UltraScale+ / UltraScale, Virtex-7 and newer technologies</p> <p>The design plan contains partition definitions with logic assigned to them so that you can help guide resource placement for the design. You can also assign I/Os to specific FPGA device pin numbers.</p>
.fdc	Synopsys FPGA Design Constraint	<p>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, on page 147.</p>

Extension	File	Description
.ini	Configuration and Initialization	Governs the behavior of the synthesis tool. You normally do <i>not</i> 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 587 .
		On the Windows 7 platforms, the ini file is in the C:\Users\userName\AppData\Roaming\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).
.ncf	Xilinx constraints file	Contains Xilinx constraints that can be translated to the sdc format and used to drive synthesis.
.nrf	Netlist Restructure	<i>Synplify Premier</i> Displays the netlist restructure file view along with the HDL Analyst RTL view to perform bit slicing. For more information, see Netlist Restructure File, on page 658 .
.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	Intel constraints file	Contains Intel 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.

Extension	File	Description
.sfp	Design Plan	<p><i>Synplify Premier with Design Planner</i> <i>Intel Stratix IV and Cyclone II and older; Xilinx-6 and older technologies</i></p> <p>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. For example:</p> <ul style="list-style-type: none"> • 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 (Intel DSPs) • Pin assignments: assignment of I/Os to specific FPGA device pin numbers
.sv	Source files (Verilog)	<p>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 186, Chapter 1, Verilog Language Support, and Chapter 2, SystemVerilog Language Support. For information about using VHDL and Verilog files together in a design, see Using Mixed Language Source Files, on page 81 of the <i>User Guide</i>.</p>
.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.

Extension	File	Description
.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 185 , Chapter 3, VHDL Language Support , and Chapter 4, VHDL 2008 Language Support for details. For information about using VHDL and Verilog files together in a design, see Using Mixed Language Source Files, on page 81 of the <i>User Guide</i> .
.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 186 , Chapter 1, Verilog Language Support , and Chapter 2, SystemVerilog Language Support . For information about using VHDL and Verilog files together in a design, see Using Mixed Language Source Files, on page 81 of the <i>User Guide</i> .

HDL Source Files

*Synplify Pro, Synplify Premier
Synplify tool must be all in one format*

The HDL source files for a project can be in either VHDL (.vhd), Verilog (.v), or SystemVerilog (.sv) 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;
```

For more information about the VHDL language, and the synthesis commands and attributes you can include, see [Chapter 3, VHDL Language Support](#) and [Chapter 4, VHDL 2008 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 1, Verilog Language Support](#) and [Chapter 2, 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.

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 several 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 the *Appendices* of the *Reference Manual*.

- The open verification library is automatically included in the FPGA product installation. When using your own open verification library, follow the recommendation described in [Open Verification Library \(Verilog\), on page 188](#).
- Technology-independent libraries that contain common components. You can have your own library or use the one Synopsys provides. This library is a Verilog library of common logic elements, much like the Synopsys® GTECH component library. See [The Generic Technology Library, on page 188](#) for a description of this library.
- An ASIC Library Data Format file (.lib) is the technology library file that contains information about the functionality of each standard cell, its input capacitance, fanout, and timing information. For the synthesis flow to understand the instantiated or mapped ASIC primitives in the HDL, you would need to translate the functionality of the standard cell to equivalent synthesizable Verilog/VHDL definitions. To do this, you can use the lib2syn executable. For details, see [ASIC Library Files, on page 189](#).

Open Verification Library (Verilog)

The open verification library is automatically included in the FPGA product installation. If you use your own version of the open verification library, then it is recommended that you disable loading the default synovl library to avoid any conflicts between the two libraries. To do this, set the `-disable_synovl` environment variable to 1. For example:

```
#in bash  
export disable_synovl=1  
  
#in csh  
setenv disable_synovl 1
```

When the default synovl library is disabled, the following message is generated in the log file: @N:::Open Verification Library which is part of tool installation, is being disabled by option "disable_synovl".

The 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 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 generic technology library.

ASIC Library Files

An ASIC Library Data Format (.lib) is the technology library file that contains information about the functionality of each standard cell, its input capacitance, fanout, and timing information.

For the synthesis flow to understand the instantiated or mapped ASIC primitives in the HDL, you would need to manually translate the functionality of the standard cell to equivalent synthesizable Verilog/VHDL definitions. This .lib file conversion is not automated in the synthesis flow. This means that the tool will not automatically translate .lib files into corresponding and equivalent synthesizable Verilog/VHDL definitions.

However, you can use the lib2syn executable to facilitate this conversion process. The lib2syn.exe executable generates equivalent synthesizable Verilog/VHDL definitions for the cells defined in the input .lib file. You can find this executable at these locations:

- Windows: *installDirectory/bin/lib2syn.exe*
- Linux: *installDirectory/bin/lib2syn*

The executable can be run as shown in these examples:

- For Verilog output: lib2syn.exe test.lib -ovm a.vm -logfile test_lib2syn.log
- For VHDL output: lib2syn.exe test.lib -ovhm a.vhm -logfile test_lib2syn.log

The tool supports the Synopsys GTECH library flow by default, so you do not need the .lib file equivalent synthesizable Verilog/VHDL definitions for a NETLIST mapped to a GTECH library.

Note that for the synthesis flow, the lib2syn executable does not translate cells with state table definitions.

The synthesis tools do not read Synopsis Liberty format (.syn) files directly. However, there are workarounds.

- If your design has instantiated ASIC cells, do the following:
 - Get the Verilog functional files for the instantiated components.
 - Add the functional files to your project as libraries.
- If you have an ASIC library in the Liberty (.lib) or .sel format, do the following:

- Convert the ASIC library into a Verilog functional file with the lib2syn utility. The lib2syn command syntax is shown below:

installDirectory/bin/lib2syn.exe library.lib -ovm VerilogFunctionalFile

or

installDirectory/bin/lib2syn.exe library.sel -ovm VerilogFunctionalFile

- Add the functional file to your project as a library.

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 215 .
_cck.rpt	Constraint Checker Report	Checks the syntax and applicability of the timing constraints in the .fdc file for your project and generates a report (<i>projectName_cck.rpt</i>). See Constraint Checking Report , on page 218 for more information.
_compiler.linkerlog	Compiler log file for HDL source file linking	Provides details for why the VHDL and/or Verilog components in the source files were not properly linked. This file is located in the synwork directory for the implementation.
_conv.prj	Project file for new project with converted constraints	<i>Xilinx</i> Default name for the new project file generated after you run the Project->Vendor Constraints command in a design. See Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 276 in the <i>User Guide</i> for details.

Extension	File	Description
_conv.sdc	Constraints file (sdc) with translated UCF constraints	<p><i>Xilinx</i></p> <p>Contains converted UCF constraints. When you run the Project->Vendor Constraints command in a design, one file is generated for each input Xilinx constraint file. See Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 276 in the <i>User Guide</i> for details.</p>
_edif.xdc	Constraint file for Vivado place and route	<p><i>Xilinx</i></p> <p>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 1117.</p>
.est	Module area estimation file	<p><i>Synplify Premier</i></p> <p>Area estimation file that contains an area estimate for each design module. This information appears in the RTL and Technology views.</p>
_est.srr	Area estimation log	<p><i>Synplify Premier</i></p> <p>Logs area estimate information. This is the file displayed by View -> View Estimation Log.</p>
.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.

Extension	File	Description
.linkerlog	Mixed language ports/generics differences	Provides details of why the VHDL and/or Verilog components in the source files were not properly linked. This file is located in the synwork directory for the implementation. The same information is also reported in the log file.
.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 276 in the <i>User Guide</i> 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.
.pfl	Message Filter criteria	Output file created after filtering messages in the Messages window. See Updating the <i>projectName.pfl</i> file, on page 316 in the <i>User Guide</i> .

Extension	File	Description
Results file: • .edf • .edn • .vm	Vendor-specific results file	<p>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. The vendor-specific formats include the following:</p> <ul style="list-style-type: none"> • .vm for Achronix • .acf or .vqm for Intel • .edn for Lattice • .edn or .vm for Microchip • .edf format for Xilinx <p>Specify this file on the Implementation Results panel of the Implementation Options dialog box (Implementation Results Panel, on page 453).</p>
run_options.txt	Project settings for implementations	<p>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 41.</p>
.sap	Synplify Annotated Properties	<p>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 find, on page 221 and Using the Tcl Find Command to Define Collections, on page 231.</p>

Extension	File	Description
.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 Synopsys Proprietary Format. See Archive Project Command, on page 433 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	<i>Synplify Premier</i> 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).
.srr	Synthesis log file	Provides information on the synthesis run, as well as area and timing reports. See Log File , on page 199 , for more information.
.srs	Compiler output file	Output file after the compiler stage of the synthesis process. It contains an HDL-level representation of a design. This is the representation that appears graphically in an RTL view.

Extension	File	Description
synplify.ucf	UCF constraints file	<p>Xilinx</p> <p>One file generated by the synthesis tool after you run the Project->Vendor Constraints command in a design. After logic synthesis it contains the supported UCF constraints for forward-annotation. See Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 276 and in the <i>User Guide</i> for details.</p>
synplify_us.ucf	UCF constraints file	<p>Xilinx</p> <p>One file generated by the synthesis tool after you run the Project->Vendor Constraints command in a logic synthesis design. It contains the unsupported UCF constraints. See Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 276 in the <i>User Guide</i> for details.</p>
synlog folder	Intermediate technology mapping files	<p>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 284.</p>
synwork folder	Intermediate pre-mapping files	<p>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 284.</p>

Extension	File	Description
.ta	Customized Timing Report	<i>Synplify Pro, Synplify Premier</i> Contains the custom timing information that you specify through Analysis->Timing Analyst. See Analysis Menu, on page 519 , for more information.
_ta.srm	Customized mapping output file	<i>Synplify Pro, Synplify Premier</i> 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 519 for more information.
.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 231 in the <i>User Guide</i> .
.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.

Extension	File	Description
<i>vendor constraint file</i>	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 453 .
.vm .vhm	Mapped Verilog or VHDL netlist	<p>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 HDL used as input for synthesis. Specify these files on the Implementation Results dialog box (Implementation Options). See Implementation Results Panel, on page 453.</p> <p>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.</p> <p>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.</p>

Extension	File	Description
.xdc	Constraint file for Vivado place and route	<i>Xilinx</i> Contains design constraints that include Synopsys timing mode and non-timing constraints. This file is used when the synthesis tool generates a <code>vm</code> structural Verilog netlist for Vivado place and route. For more information, see Running Xilinx Vivado Place and Route, on page 1117 .

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 92](#)) 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 200
<i>Synplify Premier</i> High reliability features reported.	High Reliability Report , on page 201
Buffers added to clocks in certain supported technologies.	Clock Buffering Report , on page 202
Buffers added to nets.	Net Buffering Report , on page 202
<i>Synplify Pro, Synplify Premier</i> Compile point remapping.	Compile Point Information , on page 203
Timing results. This section of the log file begins with “START TIMING REPORT” section.	Timing Reports , on page 205
<i>Synplify Pro, Synplify Premier</i> 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.	
Resources used by synthesis mapping.	Resource Usage Report , on page 203
<i>Synplify Pro, Synplify Premier</i> Design changes made as a result of retiming.	Retiming Report, on page 204
<i>Synplify Pro, Synplify Premier</i> Design changes made as a result of gated clock conversion.	Gated Clock Conversion Report , on page 225

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.

- 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 - For details, see [Clock Pre-map Reports, on page 208](#).

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, HDL optimizations, and informational or warning messages.

High Reliability Report

Synplify Premier

This report opens the HighRel.rpt file. It contains the following information:

- Features that were applied
- Instances on which the features were applied
- Implementation status of the feature
- Voter logic results
- Sequential loop results
- Error monitoring status
- Comparator gate results
- Pipe stage results
- Informational messages

This high reliability report is also displayed in the Project Status section of Project Results view. For more information, see the [Project Status Tab](#), on page 41.

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

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

Synplify Pro, Synplify Premier

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 626](#) of 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 205](#), 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 519](#)). Their format is the same as the timing report.

Resource Usage Report

A resource usage report is added to the log file each time you compile or synthesize. The format of the 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

See [Checking Resource Usage, on page 301](#) in the *User Guide* for a brief procedure on using the report to check for overutilization.

Retiming Report

Synplify Pro, Synplify Premier

Whenever retiming is enabled, 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.

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	<i>Synplify Pro, Synplify Premier</i> Timing analysis file that contains timing information based on the parameters you specify in the stand-alone Timing Analyst (Analysis->Timing Analyst).
<i>designName</i> _async_clk.rpt.scv	<i>Synplify Pro, Synplify Premier</i> Asynchronous clock report file that is generated when you enable the related option in the stand-alone Timing Analyzer (Analysis->Timing Analyst). 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 213 for details on this report.

The timing reports in the .srr/.htm and .ta files have the following sections:

- [Timing Report Header, on page 206](#)
- [Performance Summary, on page 206](#)
- [Clock Pre-map Reports, on page 208](#)
- [Clock Relationships, on page 211](#)
- [Interface Information, on page 212](#)
- [Asynchronous Clock Report, on page 213](#)

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.

```
UUU55  
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 [ON] This timing report estimates place and route data. Please look at  
00065 [ON] Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF  
00066
```

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 455](#), for details.

Performance Summary

The Performance Summary section of the timing report lists 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.

The Performance Summary lists the following information for each clock in the system:

design:

Performance Summary Column	Description
Starting Clock	Clock at the start point of the path.
Requested/Estimated Frequency	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 Period	Target frequency goal /estimated value after synthesis. See Cross-Clock Path Timing Analysis , on page 211 for information on how cross-clock path slack is reported.
Slack	Target clock period/estimated value after synthesis.
Clock Type	Difference between estimated and requested period. See Cross-Clock Path Timing Analysis , on page 211 for information on how cross-clock path slack is reported.
Clock Group	The type of clock: inferred, declared, derived or system. For more information, see Clock Types , on page 207 .
	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.

- Derived Clocks

These are clocks that the synthesis tool identifies from a clock divider/multiplier such as DCM.

- System Clock

The system clock is the delay for the combinational 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.

Paths to/from black boxes are timed by the system clock. Add the black-box timing constraints. See [syn_black_box, on page 149](#) for the black box source code directives.

Clock Pre-map Reports

The following clock reports are generated during pre-map.

- [Clock Summary, on page 209](#)
- [Clock Load Summary, on page 209](#)
- [Clock Optimization Report, on page 210](#)

Clock Summary

Here is an example of the pre-map Clock Summary report.

Clock Summary						
Level	Start Clock	Requested Frequency	Requested Period	Clock Type	Clock Group	Clock Load
0 -	clk	10.0 MHz	100.000	declared	default_clkgroup	20
0 -	clk_comb	10.0 MHz	100.000	declared	default_clkgroup	5
0 -	clk_pin0	10.0 MHz	100.000	declared	default_clkgroup	1
0 -	clk_pin1	10.0 MHz	100.000	declared	default_clkgroup	1
0 -	clk_pin2	10.0 MHz	100.000	declared	default_clkgroup	1
0 -	clk_pin3	10.0 MHz	100.000	declared	default_clkgroup	1
0 -	clk_pin4	10.0 MHz	100.000	declared	default_clkgroup	1

Clock Load Summary

The pre-map Clock Load Summary table contains the following:

- Clock name
- Number of clock loads
- Clock source pin
- Clock load on clock pin sequential example
- Clock load on non-clock pin sequential example
- Clock load on combinatorial example

Clock Load Summary

Clock	Clock Load	Source Pin	Clock Pin Seq Example	Non-clock Pin Seq Example	Non-clock Pin Comb Example
clk	20	clk.clk(i)	i0.in1_share_reg.C	i1.i0.out.D[0]	i2.and_out0.I[1] (and)
clk_comb	5	clk_comb.clk_comb(i)	-	-	i2.and_out0.I[2] (and)
clk_pin0	1	clk_pin0.clk_pin0(i)	i1.i0.out.C	-	-
clk_pin1	1	clk_pin1.clk_pin1(i)	i1.i1.out.C	-	-
clk_pin2	1	clk_pin2.clk_pin2(i)	i1.i2.out.C	-	-
clk_pin3	1	clk_pin3.clk_pin3(i)	i1.i3.out.C	-	-
clk_pin4	1	clk_pin4.clk_pin4(i)	i1.i4.out.C	-	-

Clock Optimization Report

This is an example of the pre-map Clock Optimization report. A table is provided with information for both the Non-Gated/Non-Generated Clocks and Gated/Generated Clocks.

***** START OF PREMAP CLOCK OPTIMIZATION REPORT *****

5 non-gated/non-generated clock tree(s) driving 15 clock pin(s) of sequential element(s)
 5 gated/generated clock tree(s) driving 5 clock pin(s) of sequential element(s)
 3 instances converted, 5 sequential instances remain driven by gated/generated clocks

Non-Gated/Non-Generated Clocks				
Clock Tree ID	Driving Element	Drive Element Type	Fanout	Sample Instance
ClockId_0_5	clk_pin4	port	1	i1.i4.out
ClockId_0_6	clk_pin3	port	1	i1.i3.out
ClockId_0_7	clk_pin2	port	1	i1.i2.out
ClockId_0_8	clk_pin1	port	1	i1.i1.out
ClockId_0_9	clk_pin0	port	1	i1.i0.out
ClockId_0_10	clk	port	10	i0.outb

Gated/Generated Clocks					
Clock Tree ID	Driving Element	Drive Element Type	Unconverted Fanout	Sample Instance	Explanation
ClockId_0_0	i2.and_out4.OUT	and	1	reg4.out	Multiple clocks on instance
ClockId_0_1	i2.and_out3.OUT	and	1	reg3.out	Multiple clocks on instance
ClockId_0_2	i2.and_out2.OUT	and	1	reg2.out	Multiple clocks on instance
ClockId_0_3	i2.and_out1.OUT	and	1	reg1.out	Multiple clocks on instance
ClockId_0_4	i2.and_out0.OUT	and	1	reg0.out	Multiple clocks on instance

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 211](#) 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 297](#). For tips on using clock groups, see [Defining Other Clock Requirements, on page 261](#) 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.

$$\text{scale} = (\text{minimum time period } - (\text{-current slack})) / \text{minimum time period}$$

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.

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.

Port parameter	Description
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

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 (<i>ta</i>) 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 <i>ta</i>
Source Clock	Start clock.
Destination Clock	End clock.

Column	Description
Data Start Pin	Sequential device output pin at start of path.
Data End Pin	Setup check pin at destination.

async_clkpt.csv									
A	B	C	D	E	F	G	H	I	
Index	Path Delay	Logic Levels	Types	Route Delay	Source Clock	Destination Clock	Data Start Pin	Data End Pin	
1	1	1.533	1 LUT1_L	0.632	Clock_A	Clock_B	reg_A.Q	reg_B.D	
2	2	2.176	1 LUT1_L	0.884	Clock_B	Clock_C	reg_B.Q	reg_C.D	
4									

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, or DSPs.

This report generates technology-specific area information that 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.

Intel Hierarchical Area Report

```
##### START OF AREA REPORT #####
Part:10AS048E2FE29-2SP (Intel)
-----
##### Utilization report for Top level view: eight_bit_uc #####
-----
SEQUENTIAL ATOMS
Name      Total elements      Utilization      Notes
-----
REGISTERS      294          100 %
=====
Total SEQUENTIAL ATOMS in the block eight_bit_uc: 294 (40.50 % Utilization)

COMBINATIONAL ATOMS
Name      Total elements      Utilization      Notes
-----
ATOMS          321          100 %
COMB ARITHMETIC MODE      27          100 %
=====
Total COMBINATIONAL ATOMS in the block eight_bit_uc: 348 (47.93 % Utilization)

DSPS
Name      Total elements      Utilization      Notes
-----
MACs          1          100 %
=====
Total DSPS in the block eight_bit_uc: 1 (0.14 % Utilization)
```

```
#####
# Utilization report for cell: INS_ROM #####
Instance path: eight_bit_uc.INS_ROM
=====

COMBINATIONAL ATOMS
Name      Total elements    Utilization    Notes
-----
ATOMS      37                11.5 %        =====

Total COMBINATIONAL ATOMS in the block eight_bit_uc.INS_ROM: 37 (5.10 % Utilization)

... (All modules of the design are listed similarly.)

##### END OF AREA REPORT #####]
```

Xilinx Hierarchical Area Report

```
##### START OF AREA REPORT #####[
```

```
Part:XCVU3P-FFVC1517-1-E-EVAL (Xilinx)
```

```
#####
# Utilization report for Top level view: eight_bit_uc #####
=====
```

SEQUENTIAL ELEMENTS				
Name	Total elements	Total area	Utilization	Notes
REGISTERS	268	268	100 %	

```
=====
Total SEQUENTIAL ELEMENTS in the block eight_bit_uc: 268 (40.54 % Utilization)
```

COMBINATIONAL LOGIC				
Name	Total elements	Total area	Utilization	Notes
LUTS	318	318	100 %	
Shared LUTS	76	4294967258	100 %	
DSP48	1	1	100 %	

```
=====
Total COMBINATIONAL LOGIC in the block eight_bit_uc: 395 (59.76 % Utilization)
```

Distributed Memory				
Name	Total elements	Total area	Utilization	Notes
Distributed RAM	3	12	100 %	

```
=====
Total Distributed Memory in the block eight_bit_uc: 3 (0.45 % Utilization)
```

IO PADS				
Name	Total elements	Total area	Utilization	Notes

```
-----  
PADS      26          26      100 %  
=====
```

```
Total IO PADS in the block eight_bit_uc: 26 (3.93 % Utilization)
```

Summary Table

Name	Total elements	Total area	Utilization	Notes
LUT Site	321	292		

```
-----  
=====
```

```
##### Utilization report for cell: INS_ROM #####  
Instance path: eight_bit_uc.INS_ROM  
=====
```

COMBINATIONAL LOGIC

Name	Total elements	Total area	Utilization	Notes
LUTS	24	24	7.55 %	

```
-----  
=====
```

```
Total COMBINATIONAL LOGIC in the block eight_bit_uc.INS_ROM: 24 (3.63 % Utilization)
```

Summary Table

Name	Total elements	Total area	Utilization	Notes
LUT Site	24	24		

```
-----  
=====
```

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 90](#) in the *User Guide*.

This section describes the following topics:

- [Reporting Details, on page 218](#)
- [Inapplicable Constraints, on page 219](#)
- [Applicable Constraints With Warnings, on page 220](#)
- [Sample Constraint Check Report, on page 221](#)

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 .fdc 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.

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 219 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 220 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
create_clock	<ul style="list-style-type: none"> • Ports • Nets • Pins • Registers • Instantiated buffers
create_generated_clock	Clocks
define_compile_point	<ul style="list-style-type: none"> • Region • View
define_current_design	v:view

For these constraints ...

`set_false_path`
`set_multicycle_path`
`set_max_delay`

Objects must be ...

For -to or -from objects:
 • i:sequential instances
 • p:ports
 • i:black boxes
 For -through objects
 • n:nets
 • t:hierarchical ports
 • t:pins

`set_multicycle_path`

Specified as a positive integer

`set_input_delay`

- Input ports
- bidir ports

`set_output_delay`

- Output ports
- Bidir ports

`set_reg_input_delay`

Sequential instances

`set_reg_output_delay`

Applicable Constraints With Warnings

The following table lists reasons for warnings in the report file:

For these constraints ...

`create_clock`

Objects must be ...

- Ports
- Nets
- Pins
- Registers
- Instantiated buffers

`set_clock_uncertainty`

A single object. Multiple objects are not supported.

`define_compile_point`

A single object. Multiple objects are not supported.

`define_current_design`

`v:view`

For these constraints ...**Objects must be ...**

set_false_path

For -to or -from objects:

- i:sequential instances
 - p:ports
 - i:black boxes
- For -through objects:
- n:nets
 - t:hierarchical ports
 - t:pins

set_input_delay

A single object. Multiple objects are not supported.

set_output_delay

A single object. Multiple objects are not supported.

set_reg_input_delay

A single object. Multiple objects are not supported.

set_reg_output_delay

Sample Constraint Check Report

The following is a sample report generated by constraint checking:

```
# Synopsys Constraint Checker, version maprc, Build 1138R, built Jun 7 2016
# Copyright (C) 1994-2016, Synopsys, Inc.

# Written on Fri Jun 7 09:42:22 2016
##### 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

```
set_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

```
set_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

```
set_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]
```

```

set_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]

set_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]

set_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]

set_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]

set_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]

```

```
set_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]

set_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]

set_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]

set_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]

set_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]

set_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]
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

Synplify Pro, Synplify Premier

When using the Clock Conversion option, 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 828](#).

CHAPTER 6

RAM and ROM Inference

This chapter provides guidelines and Verilog or VHDL examples for coding RAMs for synthesis. It covers the following topics:

- [Guidelines and Support for RAM Inference](#), on page 228
- [Automatic RAM Inference](#), on page 229
- [Block RAM Inference](#), on page 233
- [LUTRAM Inference](#), on page 252
- [RAM Inference with Control Signals](#), on page 257
- [Distributed RAM Inference](#), on page 264
- [Asymmetric RAM Inference](#), on page 269
- [Asymmetric RAM Inference as a Black Box Model](#), on page 296
- [Byte-Enable RAM Inference](#), on page 302
- [UltraRAM Block Inference](#), on page 305
- [Initial Values for RAMs](#), on page 310
- [RAM Instantiation with SYNCORE](#), on page 326
- [ROM Inference](#), on page 327

Guidelines and Support for RAM Inference

There are two methods to handle RAMs: instantiation and inference. Many FPGA families provide technology-specific RAMs that you can instantiate in your HDL source code. The software supports instantiation, but you can also set up your source code so that it infers the RAMs. The following table sums up the pros and cons of the two approaches.

Inference in Synthesis	Instantiation
Advantages	Advantages
Portable coding style Automatic timing-driven synthesis No additional tool dependencies	Most efficient use of the RAM primitives of a specific technology Supports all kinds of RAMs
Limitations	Limitations
Glue logic to implement the RAM might result in a sub-optimal implementation Can only infer synchronous RAMs No support for address wrapping Pin name limitations means some pins are always active or inactive	Source code is not portable because it is technology-dependent Limited or no access to timing and area data if the RAM is a black box Inter-tool access issues, if the RAM is a black box created with another tool

You must structure your source code correctly for the type of RAM you want to infer. The following table lists the supported technology-specific RAMs that can be generated by the synthesis tool.

RAM Type	Achronix	Intel	Lattice	Microchip	Xilinx
Single Port	x	x	x	x	x
Dual Port	x	x	x	x	x
True Dual Port	x	x	x	x	x
Asymmetric	x	x			x
Byte Enable					x
UltraRAM					x

Automatic RAM Inference

Instead of instantiating synchronous RAMs, you can let the synthesis tools automatically infer them directly from the HDL source code and map them to the appropriate technology-specific RAM resources on the FPGA. This approach lets you maintain portability.

Here are some of the advantages offered by the inference approach:

- The tool automatically infers the RAM from the HDL code, which is technology-independent. This means that the design is portable from one technology to another without rework.
- RAM inference is the best method for prototyping.
- The tool automatically adds the extra glue logic needed to ensure that the logic is correct.
- The software automatically runs timing-driven synthesis for inferred RAMs.

For further details about RAM inference, see:

- *Block RAM Inference*, on page 233
- *LUTRAM Inference*, on page 252
- *RAM Inference with Control Signals*, on page 257
- *Distributed RAM Inference*, on page 264
- *Asymmetric RAM Inference*, on page 269
- *Asymmetric RAM Inference as a Black Box Model*, on page 296
- *Byte-Enable RAM Inference*, on page 302
- *UltraRAM Block Inference*, on page 305

Block RAM

The synthesis software can implement the block RAM it infers using different types of block RAM and different block RAM modes.

Types of Block RAM

The synthesis software can infer different kinds of block RAM, according to how the code is set up. For details about block RAM inference, see [Block RAM Inference, on page 233](#) and [RAM Attributes, on page 231](#). For inference examples, and see [Block RAM Examples, on page 239](#).

The synthesis tool can infer the following kinds of block RAM:

- Single-port RAM
- Dual-port RAM

Based on how the read and write ports are used, dual-port RAM can be further classified as follows:

- Simple dual-port
- Dual-port
- True dual-port

Supported Block RAM Modes

Block RAM supports three operating modes, which determine the output of the RAM when write enable is active. The synthesis tools infer the mode from the RTL you provide. It is best to explicitly describe the RAM behavior in the code, so as to correctly infer the operating mode you want. Refer to the examples for recommended coding styles.

The block RAM operating modes are described in the following table:

Mode	When write enable (WE) is active ...
WRITE_FIRST	This is a transparent mode, and the input data is simultaneously written into memory and stored in the RAM data output (DO). DO uses the value of the RAM data input (DI). See WRITE_FIRST Mode Example, on page 240 for an example.
READ_FIRST	This mode is read before write. The data previously stored at the write address appears at the RAM data output (DO) first, and then the RAM input data is stored in memory. DO uses the value of the memory content. See READ_FIRST Mode Example, on page 241 for an example.
NO_CHANGE	RAM data output (DO) remains the same during a write operation, with DO containing the last read data. See NO_CHANGE Mode Example, on page 242 for an example.

RAM Attributes

In addition to the automatic inference by the tool, you can specify RAM inference with the `syn_ramstyle` and `syn_rw_conflict_logic` attributes. The `syn_ramstyle` attribute explicitly specifies the kind of RAM you want, while the `syn_rw_conflict_logic` attribute specifies that you want to infer a RAM, but leave it to the synthesis tools to select the kind of RAM, as appropriate.

Attribute-Based Inference of Block RAM

For block RAM, the `syn_ramstyle` attribute has a number of valid values, all of which are extensively described in the documentation. This section confines itself to the following values, which are most relevant to the discussion:

<code>syn_ramstyle</code> Value	Description
<code>block_ram</code>	Enforces the inference and implementation of a technology-specific RAM.
<code>registers</code>	Prevents inference of a RAM, and maps the RAM to flip-flops and logic.
<code>no_rw_check</code>	Does not create overhead logic to account for read-write conflicts.

<code>syn_ramstyle</code> Value	Description
<code>block_ram</code>	Enforces the inference and implementation of a technology-specific RAM.
<code>registers</code>	Prevents inference of a RAM, and maps the RAM to flip-flops and logic.

If you specify the `syn_rw_conflict_logic` attribute, the synthesis tools can infer block RAM, depending on the design. If the tool does infer block RAM, it does not insert bypass logic around the block RAM to account for read-write conflicts and prevent simulation mismatches. In this way its functionality is the same as `syn_ramstyle` with `no_rw_check`, which does not insert bypass logic either.

Specifying the Attributes

You set the attribute in the HDL source code, through the SCOPE interface or in an FPGA constraint file.

HDL Source Code

Set the attribute on the Verilog register or VHDL signal that holds the output values of the RAM. The following syntax shows how to specify the attribute in Verilog and VHDL code:

```
Verilog reg [7:0] ram_dout [127:0]
/*synthesis syn_ramstyle = "block_ram"*/;
reg [d_width-1:0] mem [mem_depth-1:0]
/*synthesis syn_rw_conflict_logic = 0*/;

VHDL attribute syn_ramstyle of ram_dout : signal is "block_ram";
```

SCOPE

For the `syn_ramstyle` attribute, set the attribute on the RAM register memory signal, `mem`, as shown below. For the `syn_rw_conflict_logic` attribute, set it on the instance or set it globally. The attributes are written out to a constraints file using the syntax described in the next section.

	Enabled	Object Type	Object	Attribute	Value	Val Type
1	<input checked="" type="checkbox"/>	<any>	i:mem[7:0]	syn_ramstyle	block_ram	string

Constraints File

In the fdc Tcl constraints file written out from the SCOPE interface, the `syn_ramstyle` attribute is attached to the register `mem` signal of the RAM, and the `syn_rw_conflict_logic` attribute is attached to the view, as shown below:

```
define_attribute {i:mem[7:0]} {syn_ramstyle} {block_ram}
define_attribute {v:mem[0:7]} syn_rw_conflict_logic {0}
```

For the `syn_rw_conflict_logic` attribute, you can also specify it globally, as well as on individual modules and instances:

```
define_global_attribute syn_rw_conflict_logic {0}
```

Block RAM Inference

Based on the design and how you code it, the tool can infer the following kinds of block RAM: single-port, simple dual-port, dual-port, and true dual-port. The details about RAM inference and setup guidelines are described here:

- [Setting up the RTL and Inferring Block RAM](#), on page 233
- [Simple Dual-Port Block RAM Inference](#), on page 235
- [Dual-Port RAM Inference](#), on page 237
- [True Dual-Port RAM Inference](#), on page 237
- [True Dual-Port Byte-Enabled RAM Inference](#), on page 238

Setting up the RTL and Inferring Block RAM

To ensure that the tool infers the kind of block RAM you want, do the following:

1. Set up the RAM HDL code in accordance with the following guidelines:
 - The RAM must be synchronous. It must not have any asynchronous control signals connected. The synthesis tools do not infer asynchronous block RAM.
 - You must register either the read address or the output.
 - The RAMs must not be too small, as the tool does not infer block RAM for small-sized RAMs. The size threshold varies with the target technology.

2. Set up the clocks and read and write ports to infer the kind of RAM you want. The following table summarizes how to set up the RAM in the RTL:

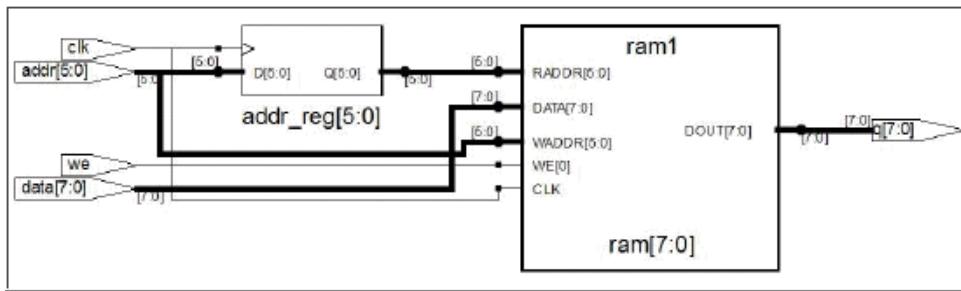
RAM	Clock	Read Ports	Write Ports
Single-port	Single clock	One; same as write	One; same as read
Simple dual-port	Single or dual clock	One dedicated read	One dedicated write
Dual-port	Single or dual clock	Two independent reads	One dedicated write
True dual-port	Single or dual clock	Two independent reads	Two independent writes

See [Dual-Port RAM Inference , on page 237](#) and [True Dual-Port RAM Inference , on page 237](#) for additional information.

For illustrative code examples, see the single-port and dual-port examples listed in [Block RAM Examples, on page 239](#).

- If needed, guide automatic inference with the `syn_ramstyle` attribute:
 - To force the inference of block RAM, specify `syn_ramstyle=blockram`.
 - To prevent a block RAM from being inferred or if your resources are limited, use `syn_ramstyle=registers`.
 - If you know your design does not read and write to the same address simultaneously, specify `syn_ramstyle=no_rw_check` to ensure that the synthesis tool does not unnecessarily create bypass logic for resolving conflicts.
- Synthesize the design.

The tool first compiles the design and infers the RAMs, which it represents as abstract technology-independent primitives like RAM1 and RAM2. You can view these RAMs in the RTL view, which is a graphic, technology-independent representation of your design after compilation:



It is important that the compiler first infers the RAM, because the tool only maps the inferred RAM primitives to technology-specific block RAM. Any RAM that is not inferred is mapped to registers. You can view the mapped RAMs in the Technology view, which is a graphic representation of your design after synthesis, and shows the design mapped to technology-specific resources.

Simple Dual-Port Block RAM Inference

Simple dual-port RAMs (SDP) are block RAMs with one port dedicated to read operations and one port dedicated to write operations. SDP RAMs offer the unique advantage of combining ports and using them to pack double the data width and address width.

The synthesis tools map SDP RAMs to RAM primitives in the architecture. A unique set of addresses, clocks, and enable signals are used for each port. The synthesis tool might also set the **RAM_MODE** property on the RAM to indicate the RAM mode.

The inference of simple dual-port RAM is dependent on the size of the address and data. The RAM must follow the coding guidelines listed below to be inferred.

- The read and write addresses must be different
- The read and write clocks can be different
- The enable signals can be different

Here is an example where the tool infers SDP RAM:

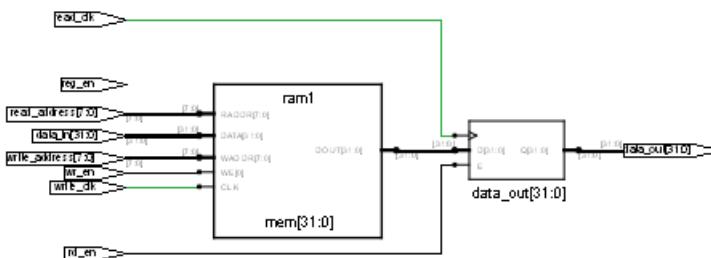
```
module Read_First_RAM (
    read_clk,
    read_address,
    data_in,
    write_clk,
    rd_en,
    wr_en,
    reg_en,
    write_address,
    data_out);

parameter address_width = 8;
parameter data_width = 32;
parameter depth = 256;
input read_clk, write_clk;
input rd_en;
input wr_en;
input reg_en;
input [address_width-1:0] read_address, write_address;
input [data_width-1:0] data_in;
output [data_width-1:0] data_out;
//wire [data_width-1:0] data_out;
reg [data_width-1:0] mem [depth -1 : 0]/* synthesis
syn_ramstyle="no_rw_check"
*/;
reg [data_width-1:0] data_out;

always @ (posedge write_clk)
if(wr_en)
    mem[write_address] <= data_in;

always @ (posedge read_clk)
if(rd_en)
    data_out <= mem[read_address];

endmodule
```



Dual-Port RAM Inference

Dual-port RAM is configured to have read and/or write operations from both ports of the RAM. One such configuration is a RAM with one port for both read and write operations and another dedicated read-only port. A unique set of addresses, clocks, and enable signals are used for each port. The synthesis tool sets properties on the RAM to indicate the RAM mode.

To infer dual-port block RAM, the RAM must follow the coding rules described below.

- The read and write addresses must be different
- The read and write clocks can be different
- The enable signals can be different

True Dual-Port RAM Inference

True dual-port RAMs (TDP) are block RAMs with two write ports and two read ports. The compiler extracts a RAM2 primitive for RAMs with two write ports or two read ports and the tool maps this primitive to TDP RAM. The ports operate independently, with different clocks, addresses and enables.

The synthesis tool also sets the `RAM_MODE` property on the RAM to indicate the RAM mode.

The compiler infers TDP block RAM based on the write processes. The implementation depends on whether the write enables use one process or multiple processes:

- When all the writes are made in one process, there are no address conflicts, and the compiler generates an nram that is later mapped to either true dual-port block RAM. The following coding 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
```

- When the writes are made in multiple processes, the software does not infer a multiport RAM unless you explicitly specify the `syn_ramstyle` attribute with a value that indicates the kind of RAM to implement, or with the `no_rw_check` value. If the attribute is not specified as such, the software does not infer an `nram`, but infers a RAM with multiple write ports. You 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 the `syn_ramstyle` attribute is specified. The writes associated with `waddr0` and `waddr1` are `we1` and `we2`, respectively.

```
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
```

True Dual-Port Byte-Enabled RAM Inference

The procedure below describes how to specify RAM where you can read/write each byte into a specific address location independently, and how to implement it as block RAM. See SolvNetPlus article 2560210, *Verilog RTL Coding Style for True Dual-Port Byte-Enabled RAM*, for an example.

- Instantiate the true dual-port RAM n number of times, where n is the number of bytes for a particular RAM address.

In the following example, `ram_dp` is instantiated twice because there are two bytes in the address:

```
ram_dp u1 (clk1, clk2, dia[7:0], addra, wea[0], doa[7:0], dib[7:0], addrb, web[0],  
dob[7:0]);  
ram_dp u2 (clk1, clk2, dia[15:8], addra, wea[1], doa[15:8], dib[15:8], addrb,  
web[1], dob[15:8]);
```

2. To map the true dual-port RAM into a block RAM, add the `syn_ramstyle="block_ram"` attribute to the true dual-port RAM module.
3. Run compile.

The RTL schematic shows two instantiations, as specified.

4. Run map.

After synthesis, check the resource utilization report to make sure that two block RAMs were inferred, as specified.

Block RAM Examples

The examples below show you how to define RAM in the HDL code so that the synthesis tools can infer block RAM. See the following for details:

- [Block RAM Mode Examples](#), on page 239
- [Single-Port Block RAM Examples](#), on page 243
- [Dual-Port Block RAM Examples](#), on page 246
- [True Dual-Port RAM Examples](#), on page 248

For details about inferring block RAM, see [Block RAM Inference](#), on page 233.

Block RAM Mode Examples

The coding style supports the enable and reset pins of the block RAM primitive. The tool supports different write mode operations for single-port and dual-port RAM. This section contains examples of how to specify the supported block RAM output modes:

- [WRITE_FIRST Mode Example](#), on page 240
- [READ_FIRST Mode Example](#), on page 241
- [NO_CHANGE Mode Example](#), on page 242

WRITE_FIRST Mode Example

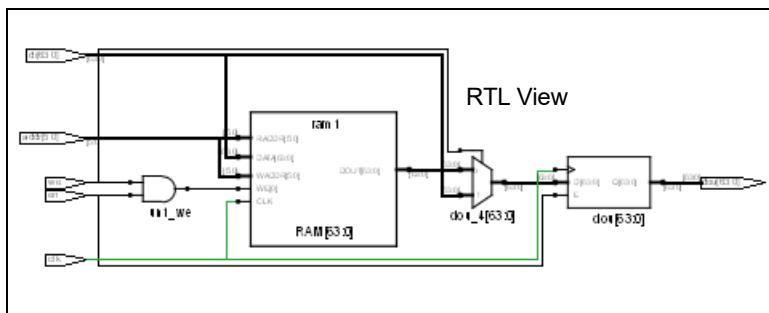
This example shows the WRITE_FIRST mode operation with active enable.

```
module v_rams_02a (clk, we, en, addr, di, dou);
    input clk;
    input we;
    input en;
    input [5:0] addr;
    input [63:0] di;
    output [63:0] dou;
    reg [63:0] RAM [63:0];
    reg [63:0] dou;

    always @ (posedge clk)
    begin
        if (en)
            begin
                if (we)
                    begin
                        RAM[addr] <= di;
                        dou <= di;
                    end
                else
                    dou <= RAM[addr];
            end
        end
    end

    always @ (posedge clk)
    if (en & we) RAM[addr] <= di;
endmodule
```

The following figure shows the RTL view of a WRITE_FIRST mode RAM with output registered. Select the Technology view to see that the RAM is mapped to a block RAM.



READ_FIRST Mode Example

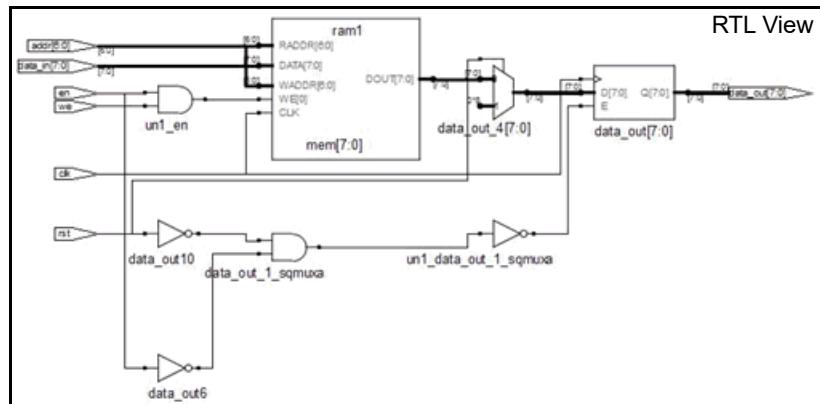
The following piece of code is an example of READ_FIRST mode with both enable and reset, with reset taking precedence:

```
module ram_test(data_out, data_in, addr, clk, rst, en, we);
output [7:0]data_out;
input [7:0]data_in;
input [6:0]addr;
input clk, en, rst, we;
reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
reg [7:0] data_out;

always@ (posedge clk)
if(rst == 1)
    data_out <= 0;
else begin
    if(en) begin
        data_out <= mem[addr];
    end
end

always @ (posedge clk)
if (en & we) mem[addr] <= data_in;
endmodule
```

The following figure shows the RTL view of a READ_FIRST RAM with inferred enable and reset, with reset taking precedence. Select the Technology view to see that the inferred RAM is mapped to a block RAM.



NO_CHANGE Mode Example

This NO_CHANGE mode example has neither enable nor reset. If you register the read address and the output address, the software infers block RAM.

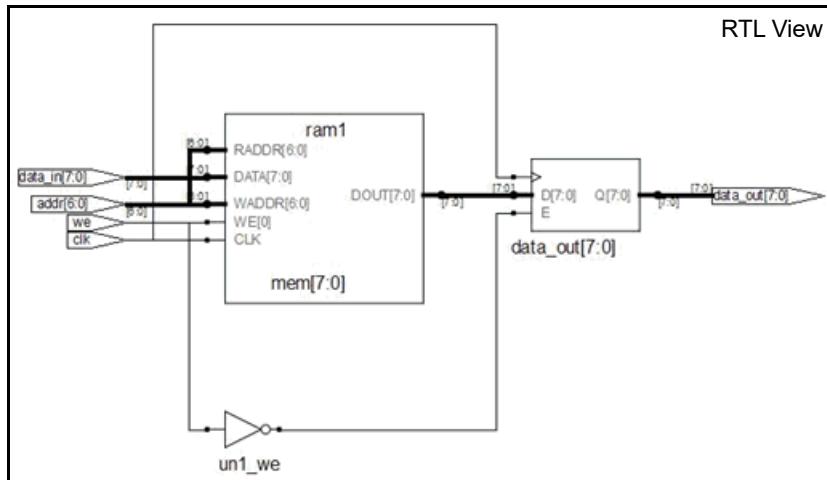
```
module ram_test(data_out, data_in, addr, clk, we);
output [7:0]data_out;
input [7:0]data_in;
input [6:0]addr;
input clk,we;
reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
reg [7:0] data_out;

always@ (posedge clk)
if (we == 1)
    data_out <= data_out;
else
    data_out <= mem[addr];

always @ (posedge clk)
if (we) mem[addr] <= data_in;

endmodule
```

The next figure shows the RTL view of a NO_CHANGE RAM. Select the Technology view to see that the RAM is mapped to block RAM.



Single-Port Block RAM Examples

This section describes the coding style required to infer single-port block RAMs. For single-port RAM, the same address is used to index the write-to and read-from RAM. See the following examples:

- [Single-Port Block RAM Examples, on page 243](#)
- [Single-Port RAM with RAM Output Registered Examples, on page 245](#)
- [Dual-Port Block RAM Examples, on page 246](#)

Single-Port RAM with Read Address Registered Example

In these examples, the read address is registered, but the write address (which is the same as the read address) is not registered. There is one clock for the read address and the RAM.

Verilog Example: Read Address Registered

```
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;
/* The array of an array register ("mem") from which the RAM is
inferred*/
reg [7:0] mem [127:0] ;
assign q = mem[read_add];

always @ (posedge clk) begin
read_add <= a;
if (we)
    /* Register RAM Data */
    mem[a] <= d;
end

endmodule

```

VHDL 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 ;

```

Single-Port RAM with RAM Output Registered Examples

In this example, the RAM output is registered, but the read and write addresses are unregistered. The write address is the same as the read address. There is one clock for the RAM and the output.

Verilog Example: Data Output Registered

```
module ram_test(q, a, d, we, clk);
    output [7:0] q;
    input [7:0] d;
    input [6:0] a;
    input clk, we;
    /* The array of an array register ("mem") from which the RAM is
    inferred */
    reg [7:0] mem [127:0];
    reg [7:0] q;

    always @ (posedge clk) begin
        q = mem[a];
        if (we)
            /* Register RAM Data */
            mem[a] <= d;
    end
endmodule
```

VHDL 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
            mem(a) <= d;
        end if;
    end if;
end process;
end rtl;

```

Dual-Port Block RAM Examples

The following example or HDL code results in simple dual-port block RAMs being implemented in supported technologies.

Verilog Example: Dual-Port RAM

This Verilog example has two read addresses, both of which are registered, and one address for write (same as a read address), which is unregistered. It has two outputs for the RAM, which are unregistered. There is one clock for the RAM and the addresses.

```

module dualportram ( q1,q2,a1,a2,d,we,clk1) ;
output [7:0]q1,q2;
input [7:0] d;
input [6:0]a1,a2;
input clk1,we;
wire [7:0] q1;
reg [6:0] read_addr1,read_addr2;
reg[7:0] mem [127:0] /* synthesis syn_ramstyle = "no_rw_check" */;
assign q1 = mem [read_addr1];
assign q2 = mem[read_addr2];

always @ ( posedge clk1) begin
read_addr1 <= a1;
read_addr2 <= a2;
if (we)
    mem[a2] <= d;
end

endmodule

```

VHDL Example: Dual-Port RAM

The following VHDL example is of READ_FIRST mode for a dual-port RAM:

```

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);

    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;

```

True Dual-Port RAM Examples

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.

Verilog Example: True Dual-Port RAM

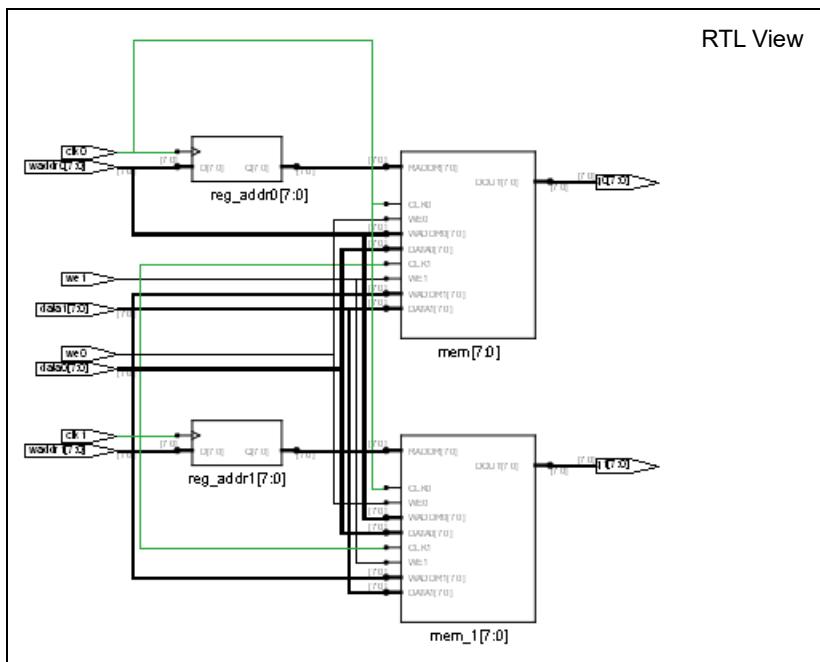
The following HDL example shows the recommended coding style for true dual-port block RAM. It is a Verilog example where the tool infers true dual-port RAM from a design with multiple writes:

```
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

always @ (posedge clk1)
begin
    reg_addr1 <= waddr1;
    if (we1)
        mem[waddr1] <= data1;
end

endmodule
```



VHDL Example: True Dual-Port RAM

The following HDL example shows the recommended coding style for true dual-port block RAM. It is a VHDL example where the tool infers true dual-port RAM from a design with multiple writes:

```

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
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
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;
```

Limitations to RAM Inference

RAM inference is only supported for synchronous RAMs.

LUTRAM Inference

Intel Technologies

The Intel technologies have LUTRAM memory components. MLAB (Memory LAB) resources are configured as LUTRAM. MLABs can be configured as single-port RAM or ROM, or simple dual-port RAM. LUTRAM writes occur on the falling edge of the clock and can be configured to have synchronous or asynchronous read.

The following procedure shows you how to set up the synthesis tool to map memory to MLABs and LUTRAMs. Note that you cannot currently map to a LUTRAM ROM, nor can you initialize asynchronous memory.

1. Start with a Stratix III design.
2. Enable the Clearbox flow option.

If this option is not enabled, the memories are mapped to ALTSYNCRAM or ALTDPRAM instead of LUTRAM.

3. Set the `syn_ramstyle` attribute to MLAB.

This automatically maps the RAM to MLAB resources, which can be configured as LUTRAMs. If you do not want to infer LUTRAM, set `syn_ramstyle` to registers.

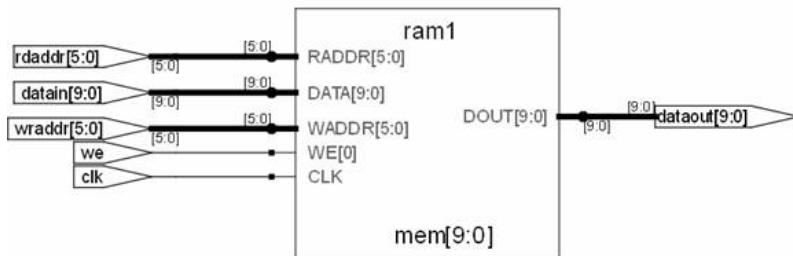
For Verilog code examples that implement LUTRAM, see [LUTRAM Examples, on page 254](#) in the *Reference Manual*.

4. Synthesize your design.

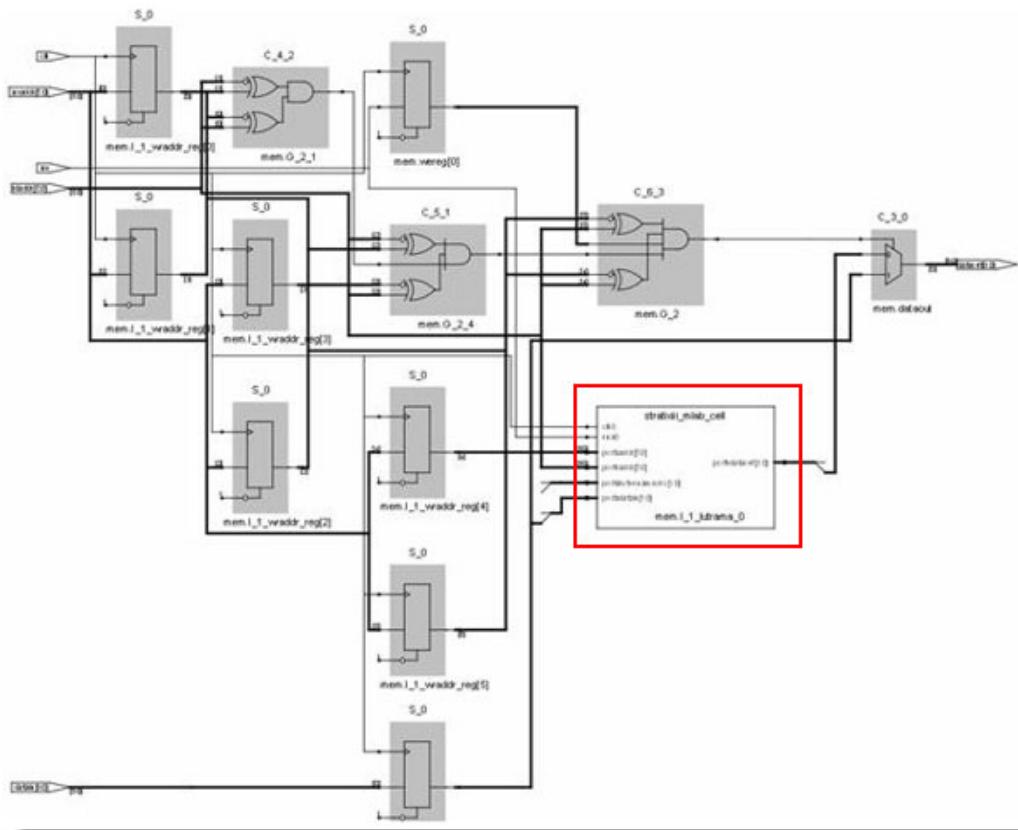
The software maps asynchronous RAMs to LUTRAM, and reports resource utilization in the log file, like this example:

```
Memory ALUTs:    10 (0% of 19000)
```

The following shows how the software maps an SDPRAM with registered output and asynchronous read to a simple dual-port RAM in the RTL view:



The following shows how the same memory is mapped in the Technology view to a `stratixiii_mlab_cell` LUTRAM component:



LUTRAM Examples

Intel Technologies

The Intel technologies have LUTRAM memory components. MLAB (Memory LAB) resources are configured as LUTRAM. MLABs can be configured as single-port RAM or ROM, or simple dual-port RAM. LUTRAM writes occur on the falling edge of the clock and can be configured to have synchronous or asynchronous read.

The following are Verilog code examples that result in memories being extracted and mapped to LUTRAM.

- [Synchronous Read Single-Port RAM, Unregistered Output](#), on page 254
- [Asynchronous Read Single-Port RAM, Unregistered Output](#), on page 255
- [Simple Synchronous Dual-Port RAM, Unregistered Output](#), on page 255
- [Simple Asynchronous Dual-Port RAM, Unregistered Output](#), on page 256

See [LUTRAM Inference](#), on page 252 for the procedure.

Synchronous Read Single-Port RAM, Unregistered Output

The following code implements a single-port LUTRAM:

```
//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:

```
//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:

```
//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

```

Simple Asynchronous Dual-Port RAM, Unregistered Output

The following code implements a simple dual-port LUTRAM:

```

//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

```

RAM Inference with Control Signals

Intel Technologies

Intel RAM blocks provide built-in functionality for control signals with enable pins. The synthesis tool extracts control signals for single-port and simple dual-port RAMs with a single common clock, and maps the logic to dedicated RAM 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.

For prerequisites to extract RAM control signals of the design, see [Guidelines for Extracting RAMs with Control Signals, on page 257](#).

Guidelines for Extracting RAMs with Control Signals

These are guidelines the tool uses for extracting RAM with control signals:

- The tool supports different clock modes as follows:

Single-port, simple dual-port, and true dual-port RAM	Input or output clock mode
Simple dual-port RAM	Read-write clock mode
True dual-port RAM	Independent clock mode

- For clock enables with read enable and write enable, the design must have a single clock.
- For true dual-port RAM, the tool supports RAM with one write and two read processes.
- If a negative level-sensitive signal is combined with an enable, the software might not extract a RAM.

Guidelines for WRITE_FIRST Mode

- The tool does not support two clock enables with read enable and write enable in WRITE_FIRST mode.
- To extract read enable in WRITE_FIRST mode, register the read enable signal. With READ_FIRST mode, the read enable is inferred 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.

Guidelines for READ_FIRST Mode

- To implement the clock enable and read enable for the controlling read operation in READ_FIRST mode, register the output of the RAM with two stage registers. The read enable controls the first register output and the clock enable controls the second register stage.
- To implement the clock enable with read and write enables in READ_FIRST mode, the output register must be enabled by both the 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.

For code examples of RAM with control signals, see [RAM with Control Signals Examples, on page 258](#).

RAM with Control Signals Examples

Intel Technologies

Intel RAM blocks provide built-in functionality for control signals with enable pins. The synthesis tool extracts control signals for single-port and simple dual-port RAMs with a single common clock, and maps the logic to dedicated RAM 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.

Here are examples that show how RAM with control signals are inferred:

- [Dual-Port RAM with Read Enable and Clock Enable, on page 259](#)

- Single-Port RAM with Clock Enable Controlling Read, on page 260
- Dual-Port RAM with Read Enable, Write Enable, and Common Clock Enable, on page 262

For details about inferring RAM with control signals, see [RAM Inference with Control Signals, on page 257](#).

Dual-Port RAM with Read Enable and Clock Enable

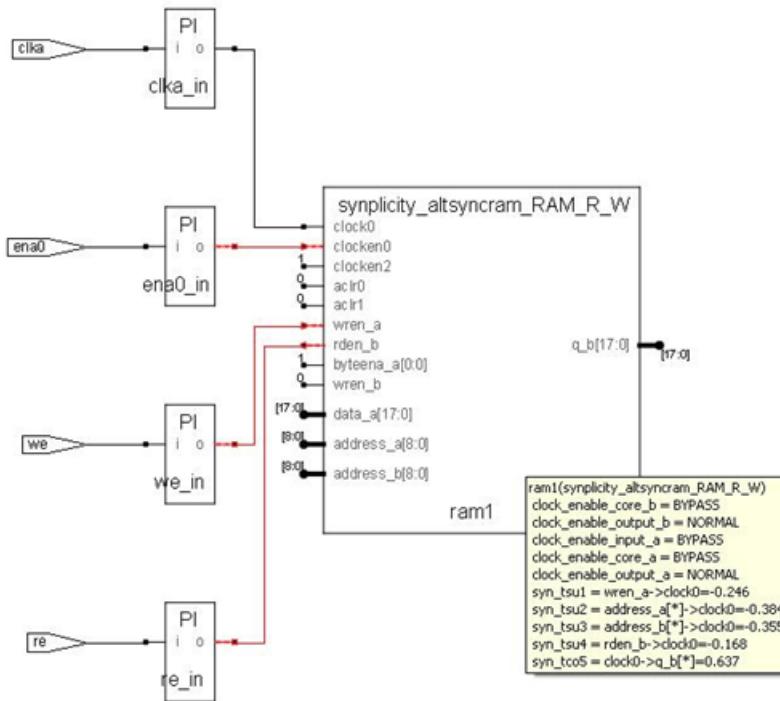
The following example shows code for a simple dual-port RAM in READ_FIRST mode with read enable and clock enable, and the altsyncram that is inferred.

```
module test (addra,addrb,clka,dinla,out1a,we,re,ena0);
    input [8:0] addra, addrb;
    input     clka;
    input we,re,ena0;
    input [17:0] dinla;
    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]] <= dinla;
    end

    always@ (posedge clka)
    begin
        if (ena0)
            out1a <= out1a_reg1;
    end
endmodule
```



Single-Port RAM 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 (addr,a,clk,din,out,we,ena);
  input [12:0] addr;
  input clk;
  input we,ena;
  input [17:0] din;
  output reg [17:0] out;
  reg [12:0] addr_reg;
  reg re_reg;
  reg [17:0] ram1 [8191:0];
endmodule
```

```

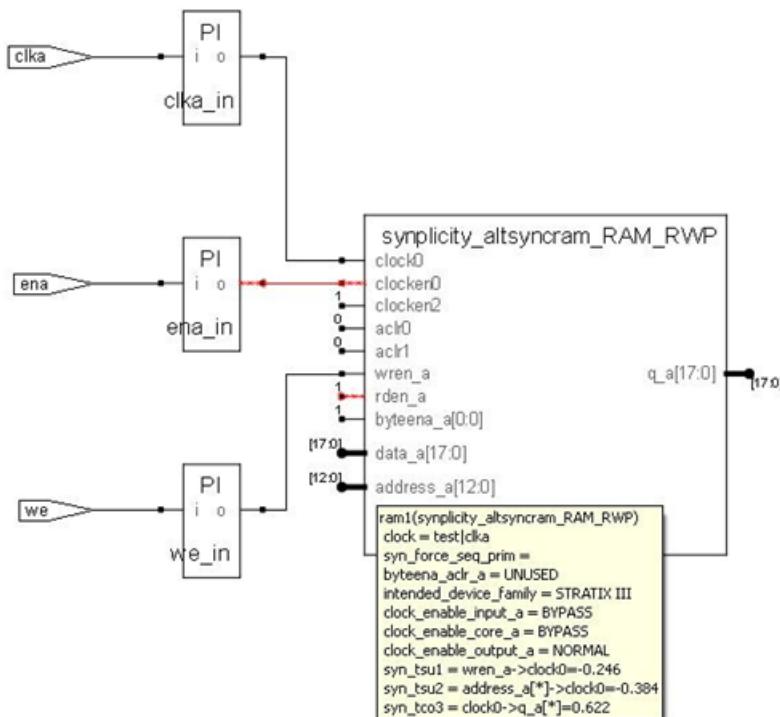
always@ (posedge clka )
begin
    if (ena)
        out1a <= ram1 [addr1a[12:0]];
end

//Write code Port A
always@ (posedge clka)
begin
    if (we)
        ram1 [addr1a[12:0]] <= din1a;
end

always@ (posedge clka)
begin
    addr1a <= addra;
end

endmodule

```



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      (addr_a,addr_b,clk_a,din_a,out_a,we,re,ena0);
parameter addr_width = 12;
parameter data_width = 36;
parameter mem_depth = 4096;

input [addr_width - 1:0] addr_a, addr_b;
input clk_a;
input we,re,ena0;
input [data_width - 1 :0] din_a;
output reg [data_width - 1 :0] out_a;
reg [addr_width - 1 :0] addr_a_reg;
reg re_reg;
reg [data_width -1:0] ram1 [mem_depth - 1:0] ;

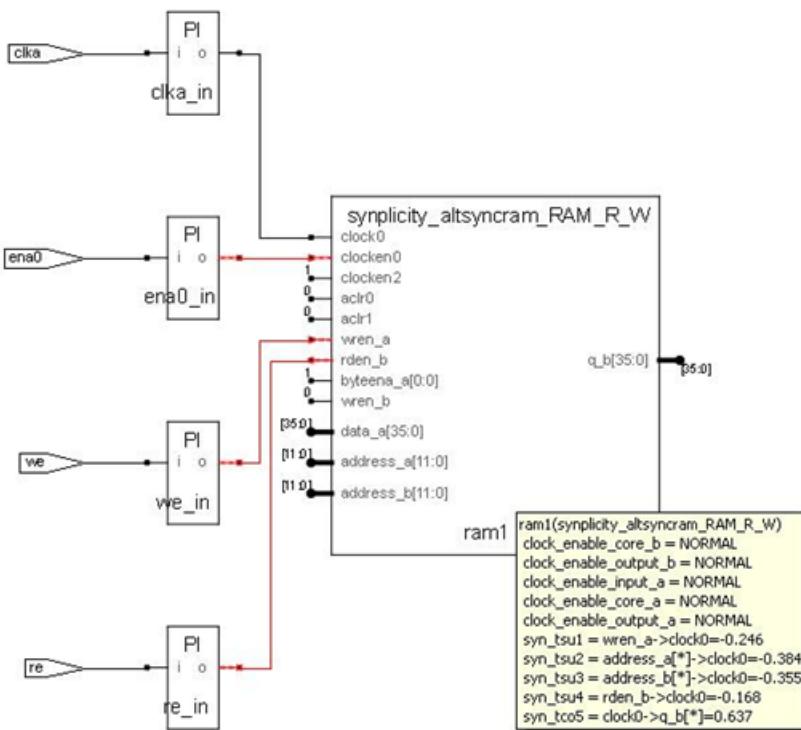
always@(posedge clk_a)
  if (re_reg & ena0)
    out_a <= ram1[addr_a_reg[addr_width - 1:0]];

//Write code Port A
always@(posedge clk_a)
begin
  if (we & ena0)
    ram1[addr_b[addr_width - 1:0]] <= din_a;
end

always@(posedge clk_a)
begin
  if (ena0)
    addr_a_reg <= addr_a;
end

always@(posedge clk_a)
begin
  if (ena0)
    re_reg <= re;
end

endmodule
```



Distributed RAM Inference

Xilinx Technologies

Distributed RAMs are inferred memories that the synthesis tools do not map to the dedicated block RAM memory resources. Instead, the tools implement them using regular lookup tables (LUTs) within a slice of the configurable logic block (CLB). Typically, distributed RAMs are used for small embedded memory blocks, like synchronous and asynchronous FIFOs, for example.

Distributed RAMs have a single clock. Reads can be asynchronous, but writes are synchronous. The tool supports the following memory types for distributed RAM:

- Single-port distributed RAM
- Dual-port distributed RAM
- Multiport distributed RAM

Attribute-Based Inference of Distributed RAM

By default, the tool implements any memories with unregistered outputs or read addresses as distributed RAM or logic. You can use the `syn_ramstyle` attribute to specify the implementation you want.

syn_ramstyle Value Description

<code>select_ram</code>	Enforces the inference and implementation of a technology-specific distributed RAM.
<code>registers</code>	Prevents inference of a RAM, and maps the inferred RAM to flip-flops and logic.

If you specify the `syn_rw_conflict_logic` attribute, the synthesis tools can infer distributed RAM, depending on the design.

For information about setting up your design to infer distributed RAM, see [Inferring Distributed RAM, on page 265](#) and [RAM Attributes, on page 231](#). For examples of distributed RAM, see [Distributed RAM Examples, on page 267](#).

Inferring Distributed RAM

Xilinx Technologies

Based on the design and how you code it, the tool can infer the following kinds of distributed RAM: single-port, dual-port, and multiport. To ensure that the tool infers the kind of RAM you want, do the following:

1. Set up the RAM HDL code in accordance with the following guidelines:
 - The RAM can be synchronous or asynchronous.
 - Do not register the read address or the output. If you add a register, the software implements block RAM, not distributed RAM.
 - To be automatically inferred, make sure the RAM is above the minimum size threshold.

You can also guide inference with the `syn_ramstyle` attribute, as described in step 3.

2. Set up the clocks and read and write ports to infer the kind of RAM you want. The following table summarizes how the RAM must be set up:

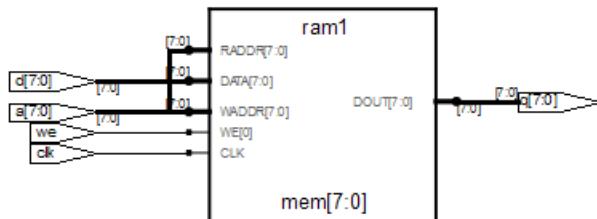
Distributed RAM	Read Ports	Write Ports
Single Port	One; same as write	One; same as read
Dual Port	One dedicated read	One dedicated write
Multiport	3 or 4 independent read ports	One shared write port

The inference of multiport distributed RAM depends on whether one or multiple write processes are used. See [True Dual-Port RAM Inference, on page 237](#) for details.

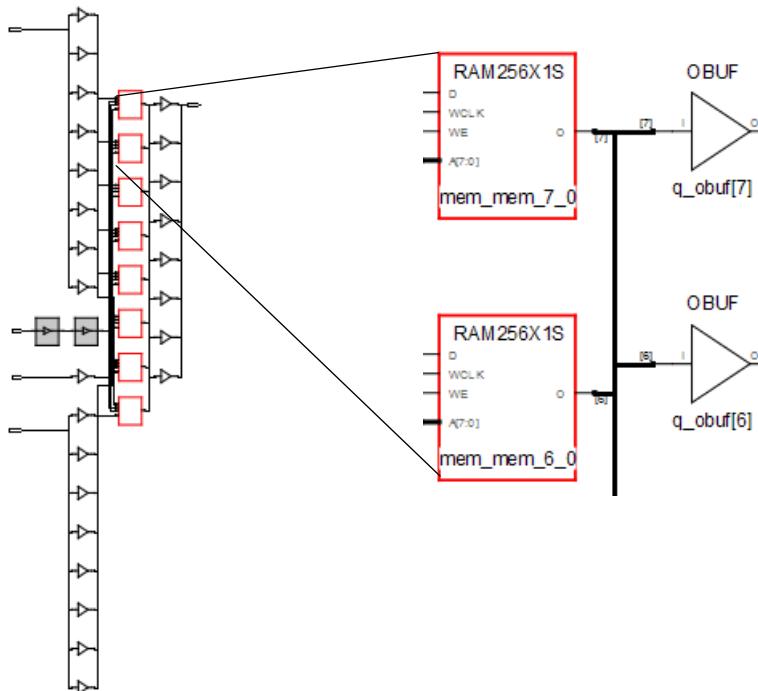
3. If needed, guide automatic inference with the `syn_ramstyle` attribute:
 - To force the inference of distributed RAM, specify `syn_ramstyle=select_ram`.
 - To prevent a distributed RAM from being inferred, use `syn_ramstyle=logic`.

4. Synthesize the design.

The tool first compiles the design and infers the RAMs, which it maps to abstract technology-independent primitives. The following figure shows an inferred RAM in the RTL view:



The tool then maps the inferred RAM primitives to technology-specific distributed RAM. The following view shows the inferred RAM from the RTL view mapped to technology-specific RAM resources in the Technology view. The RAMs are highlighted in red:



Single-Port Distributed RAM Inference

For single-port RAM, the same address is used for reading and writing operations. The tool automatically infers single-port distributed RAM in the following cases:

- The block RAM resources have been exhausted
- The RAM contains an asynchronous read port
- The `syn_ramstyle` attribute is set to `select_ram`

The RTL view shows a RAM1 primitive inferred.

Dual-Port Distributed RAM Inference

For distributed RAM, dual-port inference is the same as for block RAM. See [Dual-Port RAM Inference, on page 237](#) for details.

Multiport Distributed RAM Inference

Newer Xilinx technologies, like Virtex-5 and later, have multiport distributed RAM primitives, like RAM32M and RAM64M for example. These RAMs have four ports, generally configured as one write and three read ports. The RAM configurations follow the DRC rules provided by Xilinx.

When the compiler encounters more than two ports in the RTL, it infers an nram primitive. The nrams are mapped to RAM32M or RAM64M primitives by the mapper.

Distributed RAM Examples

Xilinx Technologies

Here are some examples of distributed RAM:

- [Single-Port Distributed RAM with Asynchronous Read Port, on page 268](#)
- [Dual-Port Distributed RAM with One Write and Two Asynchronous Reads, on page 268](#)

For details about inferring distributed RAM, see [Distributed RAM Inference, on page 264](#).

Single-Port Distributed RAM with Asynchronous Read Port

The tool implements a single-port distributed RAM with asynchronous read port for the following code:

```
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

    assign q = mem[a];
endmodule
```

Dual-Port Distributed RAM with One Write and Two Asynchronous Reads

For the following code, the tool implements a dual-port distributed RAM with one write and two asynchronous read ports:

```
module DP_Ram (ADDRA, ADDRDB, data_in1, data_in2, CLK, WEA1, WEA2,
    data_out1, data_out2);
    output [3:0] data_out1;
    output [3:0] data_out2;
    input [3:0] ADDRA;
    input [3:0] ADDRDB;
    input [3:0] data_in1;
    input [3:0] data_in2;
    input CLK, WEA1, WEA2;
    reg [3:0] mem [15:0];

    always@ (posedge CLK)
    begin
        if (WEA1)
            mem[ADDRA] = data_in1;
    end

    assign data_out1 = mem[ADDRA];
    assign data_out2 = mem[ADDRDB];
endmodule
```

Asymmetric RAM Inference

Intel and Xilinx Technologies

RAMs with different port widths for read and write operations are called asymmetric RAMs. The synthesis tool supports different port widths for read and write ports within a single block RAM primitive. The read and write widths vary by 2^{**n} . The synthesis tool infers asymmetric RAM if it is coded as a contiguous write operation.

For code examples of asymmetric RAM, see [Asymmetric RAM Examples, on page 269](#).

Asymmetric RAM Examples

The synthesis tool provides 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 block RAM primitive for the simple dual-port RAM.

Support is limited to specific coding styles of simple dual-port RAM shown in the following examples:

- [Example 1 \(Read Width > Write Width\): Coding Style 1, on page 269](#)
- [Example 1 \(Read Width > Write Width\): Coding Style 2, on page 276](#)
- [Example 2 \(Read Width < Write Width\): Coding Style 1, on page 283](#)
- [Example 2 \(Read Width < Write Width\): Coding Style 2, on page 289](#)
- [Limitations for Asymmetric RAM Inference, on page 294](#)

For details about inferring asymmetric RAM, see [Asymmetric RAM Inference, on page 269](#).

Example 1 (Read Width > Write Width): Coding Style 1

Intel and Xilinx Technologies

The following Verilog and VHDL examples implement an asymmetric port RAM where:

- Port A is 16384x2-bit write-only

- Port B is 8192x4-bit read-only
- Write first mode has no control signals on address register with different clocks. Enable is on the write port, but not the read port.

Example 1: Verilog Asymmetric RAM Coding Style 1

```
module asymmetric_ram (clkA, clkB, weA, enA, addrA, addrB, diA,  
dB);
```

```
parameter WIDTHA      = 2;  
parameter SIZEA       = 16384;  
parameter ADDRWIDTHA = 14;  
parameter WIDTHB      = 4;  
parameter SIZEB       = 8192;  
parameter ADDRWIDTHB = 13;
```

```
input                  clkA;  
input                  clkB;  
input                  weA;  
input                  enA;  
input [ADDRWIDTHA-1:0]  addrA;  
input [ADDRWIDTHB-1:0]  addrB;  
input [WIDTHA-1:0]      diA;  
output reg [WIDTHB-1:0] dB;
```

```
`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      [ADDRWIDTHB-1:0] addrB_reg;
genvar i;

always @ (posedge clkA)
```

```
begin
    if ( enA & weA)
        RAM[addrA] <= diA;
    end

always @ (posedge clkB)
begin
    addrB_reg <= addrB;
end

generate for (i = 0; i < RATIO; i = i+1)
begin: ramread
    localparam [log2RATIO-1:0] lsbaddr = i;
    always @ (posedge clkB)
    begin
        doB[(i+1)*minWIDTH-1:i*minWIDTH] <= RAM[
            {addrB_reg, lsbaddr}];
    end
end
endgenerate

endmodule
```

Example 1: VHDL Asymmetric RAM Coding Style 1

```
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 := 2;
        SIZEA       : integer := 16384;
        ADDRWIDTHA : integer := 14;
        WIDTHB      : integer := 4;
        SIZEB       : integer := 8192;
        ADDRWIDTHB : integer := 13
    );
    port (
        clkA      : in std_logic;
        clkB      : in std_logic;
        weA       : in std_logic;
        enA       : 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

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;
```

```
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);
signal ram : ramType := (others => (others => '0'));
signal addrB_reg : std_logic_vector(ADDRWIDTHB-1 downto 0);

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)
begin
    if rising_edge(clkB) then
```

```

        for i in 0 to RATIO-1 loop
            doB((i+1)*minWIDTH-1 downto i*minWIDTH) <=
                ram(conv_integer(addrB_reg &
                    conv_std_logic_vector(i,log2(RATIO))));

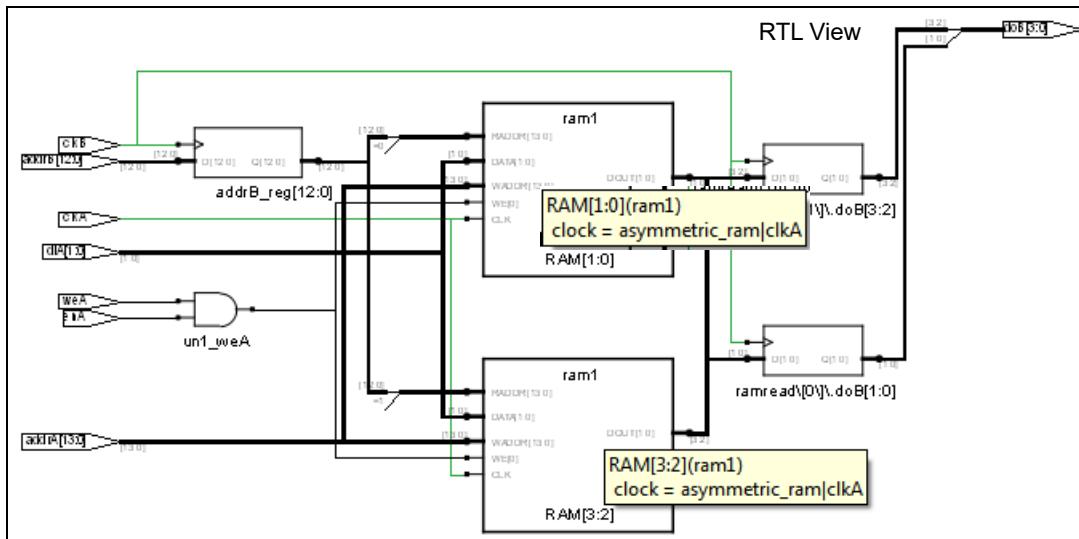
            addrB_reg <= addrB;
        end loop;

        end if;
    end process;

    end behavioral;

```

The following example shows the RTL View for Coding Style 1:



Example 1 (Read Width > Write Width): Coding Style 2

Intel and Xilinx Technologies

The following Verilog and VHDL examples implement an asymmetric port RAM where:

- Port A is 1024x2-bit write-only
- Port B is 256x8-bit read-only
- Write first mode has no control signals on address register with different clocks. Enable is on the write port, but not the read port.

Example 1: Verilog Asymmetric RAM Coding Style 2

```
module asymmetric_ram (clkA, clkB, weA, enA, addrA, addrB, diA,  
                      doB);  
  
    parameter WIDTHA      = 2;  
    parameter SIZEA       = 1024;  
    parameter ADDRWIDTHA = 10;  
    parameter WIDTHB      = 8;  
    parameter SIZEB       = 256;  
    parameter ADDRWIDTHB = 8;  
  
    input                  clkA;  
    input                  clkB;  
    input                  weA;  
    input                  enA;  
    input [ADDRWIDTHA-1:0]  addrA;  
    input [ADDRWIDTHB-1:0]  addrB;  
    input [WIDTHA-1:0]      diA;  
    output reg [WIDTHB-1:0] doB;  
  
`define max(a,b) { (a) > (b) ? (a) : (b) }  
`define min(a,b) { (a) < (b) ? (a) : (b) }
```

```
localparam maxSIZE  = `max(SIZEA, SIZEB);
localparam maxWIDTH = `max(WIDTHA, WIDTHB);
localparam minWIDTH = `min(WIDTHA, WIDTHB);
localparam RATIO    = maxWIDTH / minWIDTH;

reg      [minWIDTH-1:0]  RAM [0:maxSIZE-1];
reg      [ADDRWIDTHB-1:0] addrB_reg;

always @ (posedge clkA)
begin
  if (weA)
    begin
      RAM[addrA] <= diA;
    end
  end
end

always @ (posedge clkB)
begin
  addrB_reg <= addrB;
  doB[4*minWIDTH-1:3*minWIDTH] <= RAM[{addrB_reg, 2'd3}];
  doB[3*minWIDTH-1:2*minWIDTH] <= RAM[{addrB_reg, 2'd2}];
  doB[2*minWIDTH-1:minWIDTH]   <= RAM[{addrB_reg, 2'd1}];
  doB[minWIDTH-1:0]           <= RAM[{addrB_reg, 2'd0}];
end

endmodule
```

Example 1: VHDL Asymmetric RAM Coding Style 2

```
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 := 2;
    SIZEA       : integer := 1024;
    ADDRWIDTHA  : integer := 10;
    WIDTHB      : integer := 8;
    SIZEB       : integer := 256;
    ADDRWIDTHB  : integer := 8
);

port (
    clkA   : in std_logic;
    clkB   : in std_logic;
    weA    : in std_logic;
    enA    : 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
```

```
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;
```

```
constant minWIDTH : integer := min(WIDTHA,WIDTHB);
```

```
constant maxWIDTH : integer := max(WIDTHA,WIDTHB);
```

```
constant maxSIZE : integer := max(SIZEA,SIZEB);
```

```
constant RATIO : integer := maxWIDTH / minWIDTH;
```

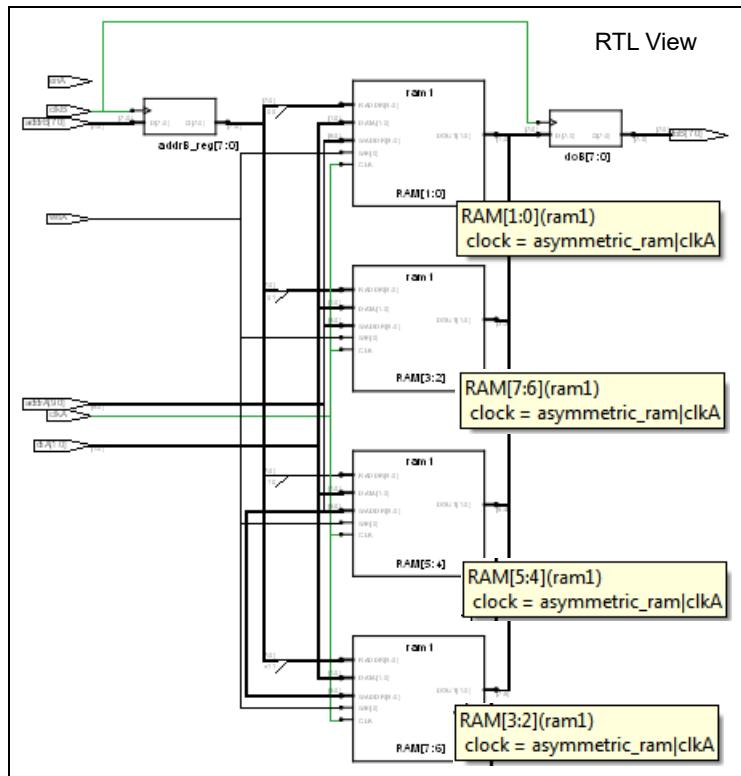
```
type ramType is array (0 to maxSIZE-1) of
    std_logic_vector(minWIDTH-1 downto 0);
signal ram : ramType := (others => (others => '0'));
signal addrB_reg : std_logic_vector(ADDRWIDTHB-1 downto 0);

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)
begin
    if rising_edge(clkB) then
        addrB_reg <= addrB;
        doB(minWIDTH-1 downto 0) <=
            ram(conv_integer(addrB_reg&conv_std_logic_vector(0,2)));
        doB(2*minWIDTH-1 downto minWIDTH) <=
            ram(conv_integer(addrB_reg&conv_std_logic_vector(1,2)));
        doB(3*minWIDTH-1 downto 2*minWIDTH) <=
```

```
ram(conv_integer(addrB_reg&conv_std_logic_vector(2,2)));\n\ndoB(4*minWIDTH-1 downto 3*minWIDTH) <=\n    ram(conv_integer(addrB_reg&conv_std_logic_vector(3,2)));\nend if;\n\nend process;\n\n\nend behavioral;
```

The following example shows the RTL View for Coding Style 2:



Example 2 (Read Width < Write Width): Coding Style 1

Intel and Xilinx Technologies

The following Verilog and VHDL examples implement 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

Example 2: Verilog Asymmetric RAM Coding Style 1

```
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;

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
```

Example2: VHDL Asymmetric RAM Coding Style 1

```
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);  
begin  
process (clkA)
```

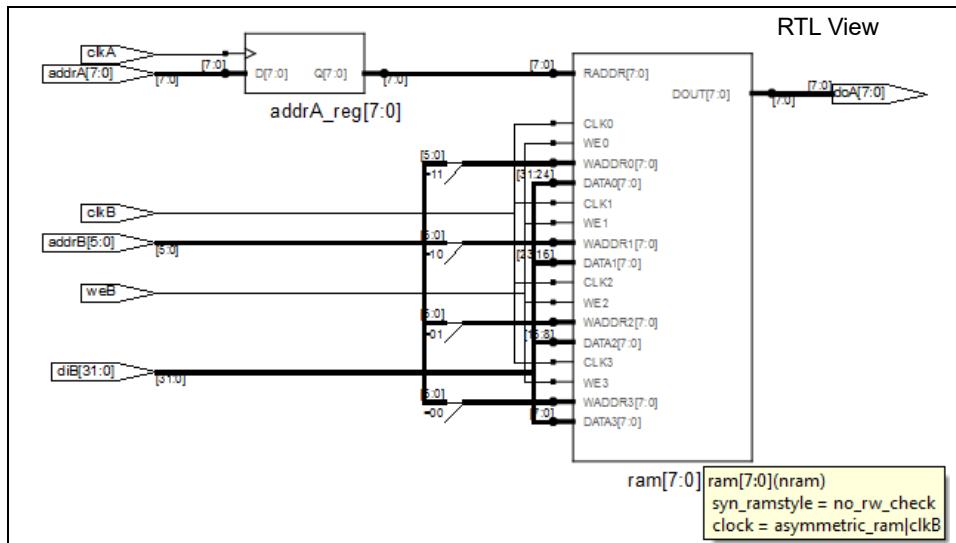
```
begin
    if rising_edge(clkA) then
        addrA_reg <= addrA;
    end if;
end process;

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:



Example 2 (Read Width < Write Width): Coding Style 2

Intel and Xilinx Technologies

The following Verilog and VHDL examples implement 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

Example 2: Verilog Asymmetric RAM Coding Style 2

```
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;
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
```

The following VHDL example implements an asymmetric port RAM:

- 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

Example 2: VHDL Asymmetric RAM Coding Style 2

```
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)
);

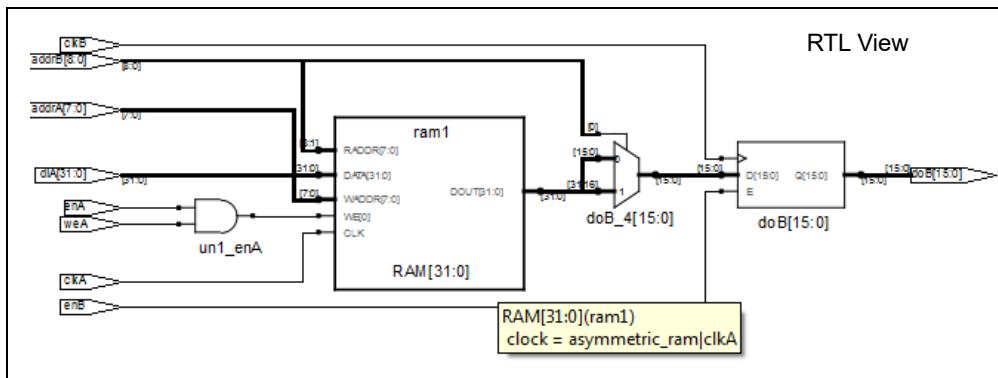
end asymmetric_ram;
```

```
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 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)));  
                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:



Limitations for Asymmetric RAM Inference

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.
- Deeper/wider RAM descriptions with random initial values that require multiple asymmetric RAM inference are not supported.

Intel-specific Limitations

Limitations specific for Intel designs include the following:

- When a RAM has two different read write clocks and read and write occurs from the same location, then the RAM output can be unknown (x).
- The enable signal is supported, but the clear signal and initial values on a Read address register are not supported.

Xilinx-specific Limitations

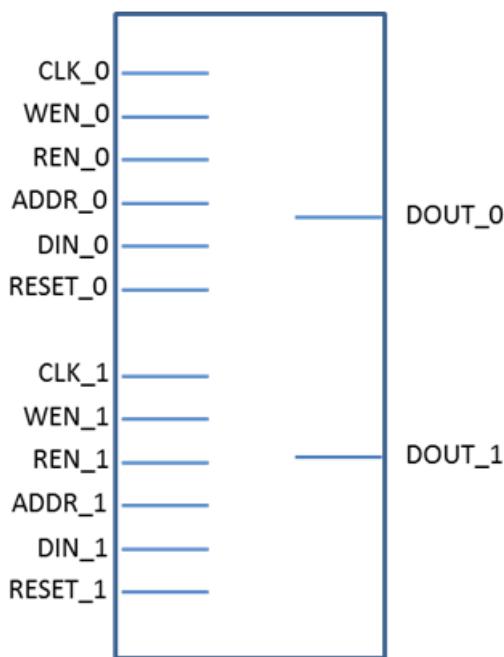
Limitations specific for Xilinx designs include in the following:

- Control signals and initial values on a Read address register are not supported.
- Possible simulation mismatch when the RAM output is not registered in WRITE_FIRST mode. Currently, the synthesis tool does not insert bypass glue logic around the RAM to prevent a simulation mismatch due to a READ_WRITE collision.

Asymmetric RAM Inference as a Black Box Model

Intel and Xilinx Technologies

True dual-port asymmetric RAM can be instantiated as black box models and inferred as synchronous RAM. To infer the asymmetric RAM, you must instantiate the following black box syn_asym_tdp_ram_model into your design.



Then, you can specify parameter values for your particular RAM design, as shown in the table below.

Asymmetric RAM Parameter Descriptions

Parameter	Type	Valid Values	Description
PORT_0_DEPTH	Integer	$2^{ADDR_0_WIDTH}$	Port 0 RAM depth
ADDR_0_WIDTH	Integer	32*	Port 0 Address width

Asymmetric RAM Parameter Descriptions

DIN_0_WIDTH	Integer	Up to 128**	Port 0 Data width
PORT_1_DEPTH	Integer	2ADDR_1_WIDTH	Port 1 RAM depth
ADDR_1_WIDTH	Integer	32	Port 1 Address width
DIN_1_WIDTH	Integer	Up to 128	Port 1 Data width
OUTPUT_REG	Integer	0 1	Inserts output data registers
READ_ADDRESS_REG	Integer	0 1	Inserts input read address registers
RESET_TYPE	String	None Synchronous Asynchronous	Indicates the reset port type
WRITE_FIRST	Integer	0 1	Indicates the RAM mode. See: <ul style="list-style-type: none">• <i>Intel-specific RAM Mode Configuration Values</i>, on page 297or• <i>Xilinx-specific RAM Mode Configuration Values</i>, on page 297

* - 12 bits. Currently, the maximum address width is limited by the depth of a single block RAM.

** - 36 bits. Currently, the maximum data width is limited by the width of the Xilinx block RAM.

Intel-specific RAM Mode Configuration Values

For Intel RAM, only WRITE_FIRST mode is supported. If a read-during-write occurs, then the new data is read out. Either READ_ADDRESS_REG must be set to 0 or WRITE_FIRST and OUTPUT_REG must both be set to 1. This ensures that the RAM is inferred in WRITE_FIRST mode with synchronous registers.

Xilinx-specific RAM Mode Configuration Values

For Xilinx RAM, both WRITE_FIRST and READ_FIRST modes are supported.

- For WRITE_FIRST mode, if a read-during-write occurs, then the new data is read out. In WRITE_FIRST mode, WRITE_FIRST and

OUTPUT_REG must both be set to 1 or READ_ADDRESS_REG must be set to 1.

- For READ_FIRST mode, if a read-during-write occurs, then the old data is read out. In READ_FIRST mode, both READ_ADDRESS_REG and WRITE_FIRST must be set to 0.

This table shows how the RAM ports can be mapped for Ports 0 and 1.

Asymmetric RAM Port Descriptions

Port	Direction	Description
CLK_0	Input	Port 0 Clock
WEN_0	Input	Port 0 Write enable. (Optional port)
REN_0	Input	Port 0 Read enable. (Optional port)
RESET_0	Input	Port 0 Reset. RESET_TYPE parameter specifies whether the reset is enabled and its type.
ADDR_0	Input	Port 0 Address. Address width is set using the ADDR_0_WIDTH parameter.
DIN_0	Input	Port 0 Data in. Data width is set using the DIN_0_WIDTH parameter.
DOUT_0	Output	Port 0 Data out. Data width is set using the DIN_0_WIDTH parameter.
CLK_1	Input	Port 1 Clock
WEN_1	Input	Port 1 Write enable. (Optional port)
REN_1	Input	Port 1 Read enable. (Optional port)
RESET_1	Input	Port 1 Reset. RESET_TYPE parameter specifies whether the reset is enabled and its type.
ADDR_1	Input	Port 1 Address. Address width is set using the ADDR_1_WIDTH parameter.
DIN_1	Input	Port 1 Data in. Data width is set using the DIN_1_WIDTH parameter.
DOUT_1	Output	Port 1 Data out. Data width is set using the DIN_1_WIDTH parameter.

Parameter Requirements

Here are the parameter requirements that must be met to infer asymmetric RAM correctly:

- By definition, the true dual-port asymmetric RAM must satisfy the following equation:

$$(\text{PORT_0_DEPTH} * \text{DIN_0_WIDTH}) = (\text{PORT_1_DEPTH} * \text{DIN_1_WIDTH})$$

- The port depths must be a power of 2.

Verilog Example: Asymmetric RAM Instantiation Model

This is a Verilog template that you can use to create an instantiation model for the asymmetric RAM.

```
syn_asym_tdp_ram_model
#(
    .PORT_0_DEPTH(PORT_0_DEPTH),
    .ADDR_0_WIDTH(ADDR_0_WIDTH),
    .DIN_0_WIDTH(DIN_0_WIDTH),

    // Port 0
    .PORT_1_DEPTH(PORT_1_DEPTH),
    .ADDR_1_WIDTH(ADDR_1_WIDTH),
    .DIN_1_WIDTH(DIN_1_WIDTH),

    // RAM Control Signals
    .OUTPUT_REG(OUTPUT_REG),
    .READ_ADDRESS_REG(READ_ADDRESS_REG),
    .RESET_TYPE(RESET_TYPE),
    .WRITE_FIRST(WRITE_FIRST)
)
Model_inst (
    .CLK_0(CLK_0),
    .WEN_0(WEN_0),
    .REN_0(REN_0),
    .RESET_0(RESET_0),
    .ADDR_0(ADDR_0),
    .DIN_0(DIN_0),
    .DOUT_0(DOUT_0),

    // Port 1
    .CLK_1(Clk_1),
    .WEN_1(WEN_1),
```

```
.RESET_1 (RESET_1),
.ADDR_1 (ADDR_1),
.DIN_1 (DIN_1),
.DOUT_1 (DOUT_1)
);
```

VHDL Example: Asymmetric RAM Instantiation Model

This is a VHDL template you can use to create an instantiation model for the asymmetric RAM.

```
Model_inst : syn_asym_tdp_ram_model
generic map (
    PORT_0_DEPTH => PORT_0_DEPTH,
    ADDR_0_WIDTH => ADDR_0_WIDTH,
    DIN_0_WIDTH => DIN_0_WIDTH,
    -- Port 0
    PORT_1_DEPTH => PORT_1_DEPTH,
    ADDR_1_WIDTH => ADDR_1_WIDTH,
    DIN_1_WIDTH => DIN_1_WIDTH,
    -- RAM Control Signals
    OUTPUT_REG => OUTPUT_REG,
    READ_ADDRESS_REG => READ_ADDRESS_REG,
    RESET_TYPE => RESET_TYPE,
    WRITE_FIRST => WRITE_FIRST
)
port map (
    CLK_0 => CLK_0,
    WEN_0 => WEN_0,
    REN_0 => REN_0,
    RESET_0 => RESET_0,
    ADDR_0 => ADDR_0,
    DIN_0 => DIN_0,
    DOUT_0 => DOUT_0,
    -- Port 1
    CLK_1 => Clk_1,
    WEN_1 => WEN_1,
    REN_1 => REN_1,
    RESET_1 => RESET_1,
    ADDR_1 => ADDR_1,
    DIN_1 => DIN_1,
    DOUT_1 => DOUT_1
);
```

Instantiation of Parameters Example

This example shows how the Verilog or VHDL templates above can actually specify parameters for the RAM. The following code snippet is an explicit WRITE_FIRST style RAM with asynchronous reset:

```
PORT_0_DEPTH(16),  
.ADDR_0_WIDTH(4),  
.DIN_0_WIDTH(32),  
// Port 0  
.PORT_1_DEPTH(32),  
.ADDR_1_WIDTH(5),  
.DIN_1_WIDTH(16),  
// RAM Control Signals  
.OUTPUT_REG(1),  
.READ_ADDRESS_REG(1),  
.RESET_TYPE("Asynchronous"),  
.WRITE_FIRST(1)
```

Intel-specific Limitations

Intel device limitations for the RAM include the following:

- For Arria 10, Arria V, Cyclone V, and Stratix V devices, only WRITE_-FIRST mode is supported across the same ports.
- For Intel devices, mixed port RAM mode is WRITE_FIRST. Glue logic is generated around the blockram for all implementations.

Xilinx-specific Limitation

Xilinx device limitations for the RAM include the following:

- Only Virtex-7 and Virtex-6 devices are supported.
- The maximum depth for the RAM is 1K and maximum width is 36 bits.
- For higher capacity devices, you must multiplex individual block RAMs.

Byte-Enable RAM Inference

Xilinx Technologies

The synthesis tools infer byte-enable RAM. Instead of using a single enable bit to write data serially to one location at a time, this feature uses multiple enable signals to read or write data to multiple locations in a block RAM simultaneously. Data can be processed in parallel, and this increases the speed and overall efficiency of the FPGA.

For:

- Code examples of byte-enable RAM, see [Byte-Enable RAM Examples, on page 302](#).
- More detailed information and examples, refer to SolvNetPlus article 030578, *Byte-Enable RAM Support*.

Byte-Enable RAM Examples

Xilinx Technologies

The synthesis tools infer byte-enable RAM. Instead of using a single enable bit to write data serially to one location at a time, this feature uses multiple enable signals to read or write data to multiple locations in a block RAM simultaneously.

For details about inferring byte-enable RAM, see [Byte-Enable RAM Inference, on page 302](#).

Byte-Enable RAM Examples

The following Verilog and VHDL examples automatically infer byte enable pins and maps it to block RAM, for code that uses an if-else-if (mutually exclusive) style:

- [Verilog Example](#)
- [VHDL Example](#)

Verilog Example

The following Verilog example infers a byte-enable RAM:

```
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
```

VHDL Example

The following VHDL example infers a byte-enable RAM:

```
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;
```

Limitations to RAM Inference

Limited support for byte-enable RAM, with one enable per byte, up to a maximum of four enables. For details, refer to [Solvnet article 030578](#), on byte-enable RAM support.

UltraRAM Block Inference

Xilinx UltraScale+ Technologies

(*Kintex UltraScale+, Virtex UltraScale+, and Zynq UltraScale+*)

The synthesis tools provide support for UltraRAM blocks, in addition to the block RAM (RAMB36E2 or RAMB18E2) for Xilinx UltraScale+ devices.

UltraRAM is a high-density memory block that has eight times the storage capacity of a block RAM. It consists of 288K bits (4Kx72 bits) of storage with two independent access ports that share a single clock. The port width is not configurable.

The tool can automatically infer an UltraRAM block for single-port and simple dual-port RAM.

For code examples, see [UltraRAM Examples, on page 305](#).

UltraRAM Examples

Xilinx UltraScale+ Technologies

(*Kintex UltraScale+, Virtex UltraScale+, and Zynq UltraScale+*)

The synthesis tools provide support for UltraRAM blocks, in addition to the block RAM (RAMB36E2 or RAMB18E2) for Xilinx UltraScale+ devices.

UltraRAM is a high-density memory block that has eight times the storage capacity of a block RAM. It consists of 288K bits (4Kx72 bits) of storage with two independent access ports that share a single clock. The port width is not configurable.

The tool can automatically infer an UltraRAM block for single-port and simple dual-port RAM.

Example: UltraRAM Inference

```
// simple dual port, no_change ram, single uram

//synthesis translate_off
`define SIMULATION 1
//synthesis translate_on
```

```
`define ADDRSIZE 12
`define DATASIZE 72

`ifdef SIMULATION
`timescale 1 ps/1 ps

module rtl_ram(din, clk, we, waddr, raddr, dout);
`else
module synth_ram(din, clk, we, waddr, raddr, dout);
`endif

input [`DATASIZE-1:0] din;
input [`ADDRSIZE-1:0] waddr;
input [`ADDRSIZE-1:0] raddr;
input clk, we;
output [`DATASIZE-1:0] dout;

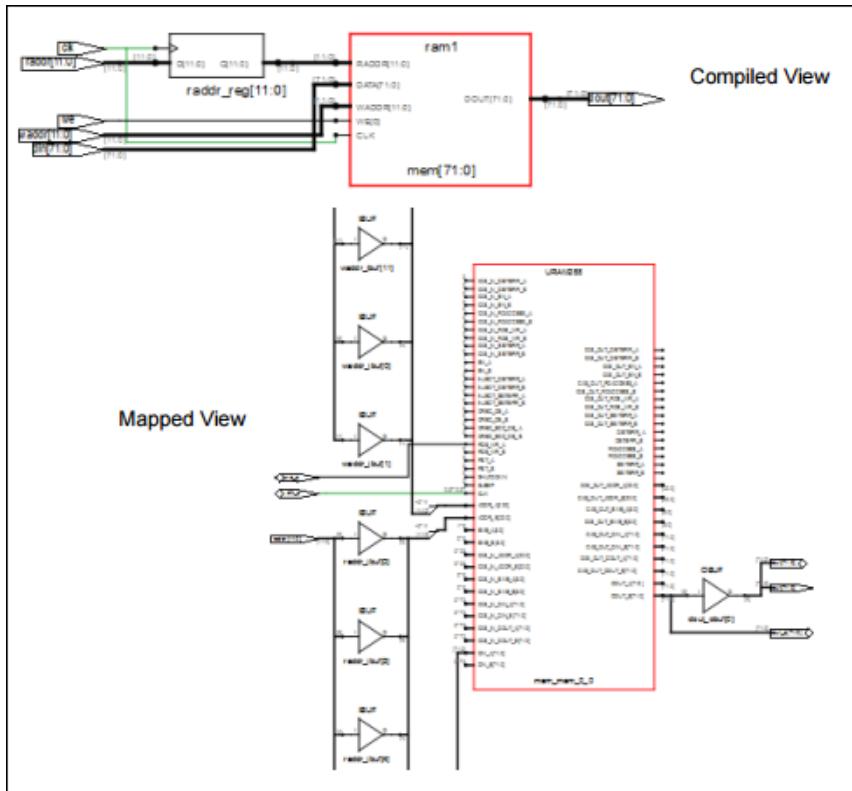
reg [`ADDRSIZE-1:0] raddr_reg;
reg [`DATASIZE-1:0] reg2;
reg [`DATASIZE-1:0] reg3;
reg [`DATASIZE-1:0]      mem [(2**`ADDRSIZE)-1:0];

always @ (posedge clk)
begin
if (we)
begin
mem[waddr] <= din;
```

```
        end  
        raddr_reg <= raddr;  
    end  
  
    assign dout = mem[raddr_reg];  
  
endmodule // top
```

This example implements a simple dual-port RAM with No Change write mode. After synthesis, a single UltraRAM block is automatically inferred.

The RAM has been mapped to the URAM288 primitive as shown in the schematic view below.



The tool can force RAM to infer the UltraRAM block, if necessary, when the `syn_ramstyle=uram` attribute is specified in the HDL source code or an FDC constraint file. You can check the timing and area reports or the netlist for this URAM288 primitive.

To disable UltraRAM block inferencing, however, specify the `syn_ramstyle=no_uram` attribute. In the Resource Utilization report below, the results of using this option show that the RAM get mapped to block RAM (RAMB36E2/RAMB18E2) instead of the UltraRAM block.

```
-----  
Resource Usage Report for synth_ram  
  
Mapping to part: xcku5p-ffva676-1-e-eval  
Cell usage:  
CARRY8      1 use  
FD          97 uses  
GND         1 use  
RAMB36E2    8 uses  
VCC         1 use  
LUT4        72 uses  
LUT6         4 uses  
  
I/O ports: 170  
I/O primitives: 170  
IBUF        97 uses  
IBUFG       1 use  
OBUF        72 uses  
  
BUFG        1 use  
I/O Register bits:           96  
Register bits not including I/Os: 1 of 442960 (0%)  
  
RAM/ROM usage summary  
Occupied Block RAM sites (RAMB36) : 8 of 480 (1%)  
  
Global Clock Buffers: 1 of 32 (3%)  
  
Total load per clock:  
  synth_ram|clk: 113
```

Limitations for UltraRAM

Limitations include the following:

- RAM cascading is not supported.
- Byte-write enable RAM is not supported.

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 HDL 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 313](#) 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

Also, see the following topics:

- [Example 1: RAM Initialization, on page 310](#)
- [Example 2: Cross-Module Referencing for RAM Initialization, on page 311](#)
- [Initialization Data File, on page 313](#)
- [Forward Annotation of Initial Values, on page 316](#)
- [Initial Values for Asymmetric RAM, on page 317](#)

Example 1: RAM Initialization

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 313](#) 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

    always @ (posedge clk)
    begin
        if (we)
        begin
            mem[addr] <= data;
        end
    end

    assign data_out = mem[addr_reg];
endmodule

```

Example 2: Cross-Module Referencing for RAM Initialization

The following example shows how a RAM using cross-module referencing (XMR) can be accessed hierarchically and initialized with the \$readmemb/\$readmemh statement which reads the values specified in the mem.txt file from the top-level design.

Example2A: XMR for RAM Initialization (Top-Level Module)

(Top-Level Module)

```

//Top
module top (input[27:0] data, input clk, we, input[10:0] addr,

```

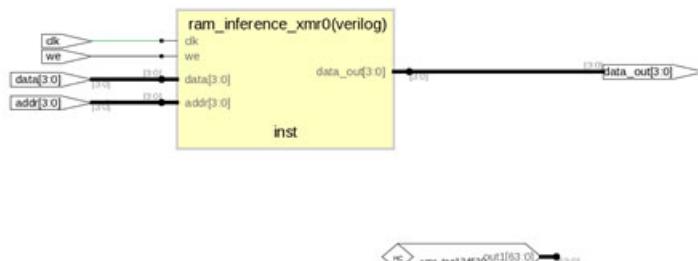
```
        output[27:0] data_out);  
  
    ram_inference ram_inst (..*);  
  
    initial  
  
    begin  
  
        $readmemb ("mem.txt", top.ram_inst.mem, 0, 10);  
  
    end  
  
endmodule
```

This code example implements cross-module referencing of the RAM block and is initialized with the \$readmemb statement in the top-level module.**Example2B: XMR for RAM Initialization (RAM)**

```
//RAM  
  
module ram_inference (input[27:0] data, input clk, 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;  
  
always @ (posedge clk)  
  
begin  
  
    addr_reg <= addr;  
  
end  
  
always @ (posedge clk)  
  
begin  
  
    if (we)  
  
    begin  
  
        mem[addr] <= data;  
  
    end  
  
end  
  
assign data_out = mem[addr_reg];
```

```
endmodule
```

Here is the code example of the RAM block to be implemented for cross-module referencing and initialized. The following shows the HDL Analyst view of a RAM module that must be accessed hierarchically to be initialized.



RAM Initialization Limitations with XMR

XMR for RAM initialization requires that the following conditions be met:

- Variables must be recognized as inferred memories.
- Cross-module referencing of memory variables cannot occur between HDL languages.
- Cross-module referencing paths must be static and cannot include an index with a dynamic value.

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 315](#)), 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).

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 315).

Binary File Format

The binary data file mem.ini that corresponds to the example in [Example 1: RAM Initialization](#), on page 310 looks like this:

```
111111111111111111111100110111 /* data for address 0 */
111111111111111111111101100111 /* data for address 1 */
1111111111111111111111111111000010
111111111111111111111111111100100001
111111111111111111111111111101110000
1111111111111111111111111111011100110
1111111111111111111111111111011100110
... /* continues until Address 1999 */
```

Hex File Format

If you use `$readmemh` instead of `$readmemb`, the hexadecimal data file for the example in [Example 1: RAM Initialization](#), on page 310 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.

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 Intel 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
:0100000003DC2
:000000001FF
```

This hex file is associated with the altsyncram in the vqm file as follows:

```
defparam mem_Z.init_file = "mem.hex";
```

Initial Values for Asymmetric RAM

Xilinx Virtex-6 and newer technologies

Asymmetric RAM can be inferred for simple dual-port RAM with content initial values and asymmetric ports, where read width is less than write width. For example, the asymmetric RAM coding style can be specified with the following conditions:

- Port A is 512x32 write-only mode (data width 32; address width 9)
- Port B is 1024x16 read-only mode (data width 16; address width 10)
- Random initial value

For more information, see the topics:

- [Verilog Example \(Read Width < Write Width\)](#)
- [VHDL Example \(Read Width < Write Width\)](#)
- [Check Results for the RAM](#)
- [Initial Value Limitations for Asymmetric RAM](#)

Verilog Example (Read Width < Write Width)

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

Example: Verilog Initial Values for Asymmetric RAM

```
//  
// Asymmetric port RAM  
// Port A is 256x8-bit read-only  
// Port B is 64x32-bit write-only  
// Write_First mode with no control signals on Address register.  
Different clocks.
```

```
//
```

```
module v_asymmetric_ram_4 (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      [WIDTHB-1:0]  readB;  
  
genvar i;  
  
reg      [minWIDTH-1:0]  RAM [0:maxSIZE-1];  
// RAM initialization  
initial
```

```
$readmemb ("mem_init_256x8.dat", RAM);  
  
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 (Read Width < Write Width)

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

Example: VHDL Initial Values for Asymmetric RAM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity asymmetric_ram_4 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;
        reA    : 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_4;
```

architecture behavioral of asymmetric_ram_4 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 =>"11111111");

signal readA : std_logic_vector(WIDTHA-1 downto 0):=
(others => '0');
signal regA  : std_logic_vector(WIDTHA-1 downto 0):=
(others => '0');
```

```
begin

process (clkA)
begin
    if rising_edge(clkA) then
        if reA = '1' then
            doA <= ram(conv_integer(addrA));
        end if;
    end if;
end process;

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;

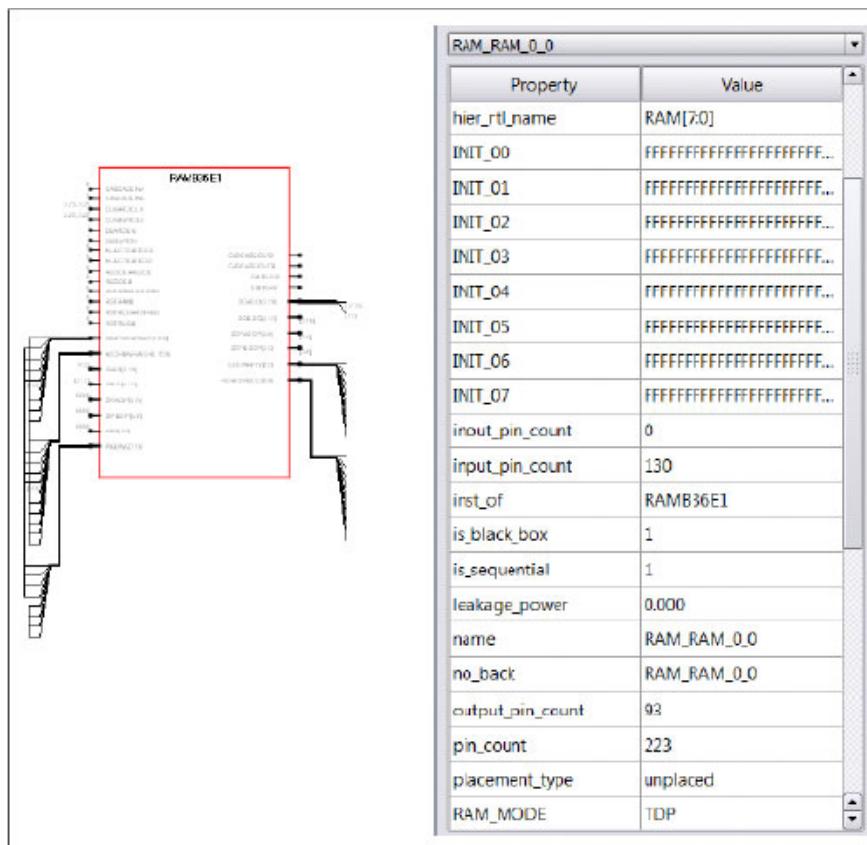
end behavioral;
```

Check Results for the RAM

After synthesizing the RAM, check the log file for startup values on the RAM as shown below:

```
Inferred asymmetric RAM 'RAM[7:0]' with read width 8 and write width 32.
| Asymmetric RAM RAM_0_0 is inferred in WRITE_FIRST mode. Simulation mismatch possible when read
Found startup values on RAM instance RAM[7:0]
Startup value RAM_0_0.INIT_00 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_01 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_02 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_03 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_04 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_05 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_06 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Startup value RAM_0_0.INIT_07 = FFFFFFFFEEEEEEEEEEEEEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEEFFFFFFEEEEE
Found startup values on RAM instance RAM[7:0]
```

Otherwise, you can right-click the RAM in the Technology view and select Properties to see the properties for the RAM as shown in the following figure:



Initial Value Limitations for Asymmetric RAM

Limitations include the following:

- Supports simple dual-port RAM coding styles for read width less than write width only.
- Deeper/wider RAM with random initial values requiring multiple asymmetric RAM inference is not supported.

See also, [Limitations for Asymmetric RAM Inference, on page 294](#).

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 [Chapter 7, SynCore IP Tool](#)

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.

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.

Verilog 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;
5'b10010 : z = 4'b0011;
default : z = 4'b0111;
endcase
end
endmodule
```

VHDL 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;
```

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
```

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
```


CHAPTER 7

SynCore IP Tool

This chapter describes the SYNCORE IP functionality that is bundled with the synthesis tool.

- [SYNCORE FIFO Compiler](#), on page 334
- [SYNCORE RAM Compiler](#), on page 365
- [SYNCORE Byte-Enable RAM Compiler](#), on page 387
- [SYNCORE ROM Compiler](#), on page 403
- [SYNCORE Adder/Subtractor Compiler](#), on page 418
- [SYNCORE Counter Compiler](#), on page 442

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 FIFO Overview](#), on page 334
- [Specifying FIFOs with SYNCore](#), on page 335
- [SYNCore FIFO Wizard](#), on page 340
- [FIFO Read and Write Operations](#), on page 349
- [FIFO Ports](#), on page 350
- [FIFO Parameters](#), on page 353
- [FIFO Status Flags](#), on page 355
- [FIFO Programmable Flags](#), on page 358

Synchronous FIFO Overview

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 349 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 350 for details.

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 corre-

sponding 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 355](#) and [FIFO Programmable Flags, on page 358](#).

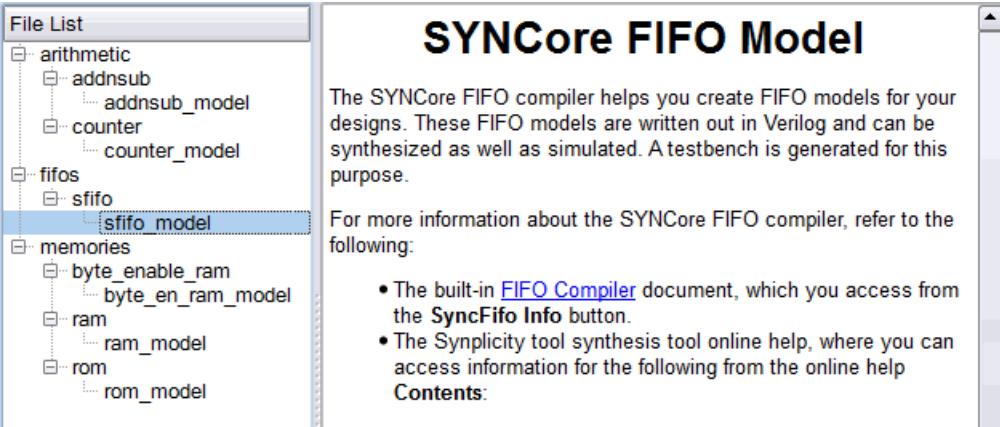
At any point in time, Data count reflects the available data inside the FIFO. In addition, you can use the Programmable Full and Programmable Empty status flags for user-defined thresholds.

Specifying FIFOs with SYNCore

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

Note: The SYNCore FIFO model uses Verilog 2001. When adding a FIFO model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a `set_option -vlog_std v2001` statement in your project file to prevent a syntax error.

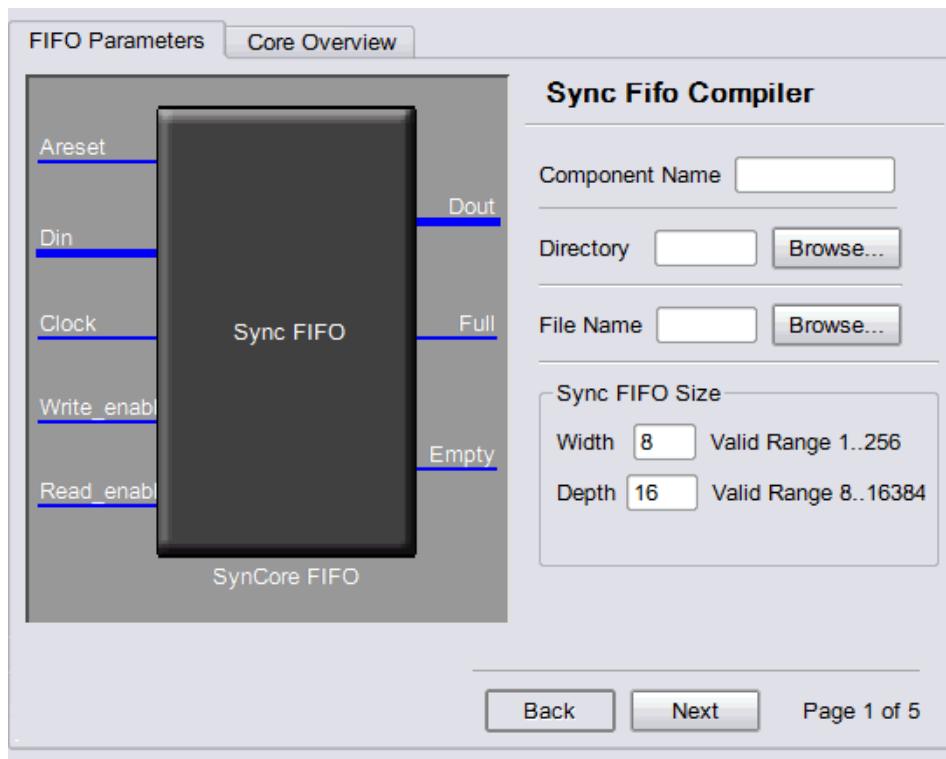
1. Start the wizard.
 - From the synthesis tool GUI, select Run->Launch SYNCore or click the Launch SYNCore icon  to start the SYNCore IP wizard.



The screenshot shows the SYNCore IP Wizard interface. On the left, a "File List" tree view shows several categories: arithmetic, addsub, counter, fifos, sfifo, memories, byte_enable_ram, ram, and rom. The "sfifo" category is expanded, and its "sfifo_model" item is selected and highlighted with a blue dotted border. On the right, the main panel displays the title "SYNCore FIFO Model". Below the title, a descriptive text states: "The SYNCore FIFO compiler helps you create FIFO models for your designs. These FIFO models are written out in Verilog and can be synthesized as well as simulated. A testbench is generated for this purpose." Further down, it says: "For more information about the SYNCore FIFO compiler, refer to the following:" followed by a bulleted list. The bulleted list includes:

- The built-in [FIFO Compiler](#) document, which you access from the [SyncFifo Info](#) button.
- The Synplicity tool synthesis tool online help, where you can access information for the following from the online help [Contents](#):

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



2. Specify the parameters you need in the five pages of the wizard. For details, refer to [Specifying SYNCore FIFO Parameters, on page 338](#).

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

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

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

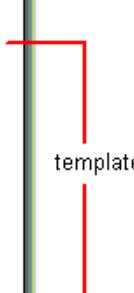
The FIFO generated is a synchronous FIFO with symmetric ports and with the same clock controlling both the read and write operations. Data is written or read on the rising edge of the clock. All resets are synchro-

nous with the clock. All edges (clock, enable, and reset) are considered positive.

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

You can now close the SYNCORE wizard.

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



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

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

5. Edit the template port connections so that they agree with the port definitions in your top-level module as shown in the example below (the updated connection names are shown in red). You can also assign a unique name to each instantiation.

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

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

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

Specifying SYNCore FIFO Parameters

The following elaborates on the parameter settings for SYNCore FIFOs. The status, handshaking, and programmable flags are optional. For descriptions of the parameters, see [SYNCore FIFO Wizard, on page 340](#).

1. Start the SYNCore wizard, as described in [Specifying FIFOs with SYNCore, on page 335](#).
2. Do the following on page 1 of the FIFO wizard:
 - In Component Name, specify a name for the FIFO. Do not use spaces.
 - In Directory, specify a directory where you want the output files to be written. Do not use spaces.
 - In Filename, specify a name for the Verilog output file with the FIFO specifications. Do not use spaces.

- Click Next. The wizard opens another page where you can set parameters.
3. For a FIFO with no status, handshaking, or programmable flags, use the default settings. You can generate the FIFO, as described in [Specifying FIFOs with SYNCORE, on page 335](#).
4. To set an almost full status flag, do the following on page 2 of the FIFO wizard:
- Enable Almost Full.
 - Set associated handshaking flags for the signal as desired, with the Overflow Flag and Write Acknowledge options.
 - Click Next when you are done.
5. To set an almost empty status flag, do the following on page 3:
- Enable Almost Empty.
 - Set associated handshaking flags for the signal as desired, with the Underflow Flag and Read Acknowledge options.
 - Click Next when you are done.
6. To set a programmable full flag, do the following:
- Make sure you have enabled Full on page 2 of the wizard and set any handshaking flags you require.
 - Go to page 4 and enable Programmable Full.
 - Select one of the four mutually exclusive configurations for Programmable Full on page 4. See [Programmable Full, on page 359](#) or details.
 - Click Next when you are done.
7. To set a programmable empty flag, do the following:
- Make sure you have enabled Empty on page 3 of the wizard and set any handshaking flags you require.
 - Go to page 5 and enable Programmable Empty.
 - Select one of the four mutually exclusive configurations for Programmable Empty on page 5. See [Programmable Empty, on page 361](#) or details.

You can now generate the FIFO and add it to the design, as described in [Specifying FIFOs with SYNCORE, on page 335](#).

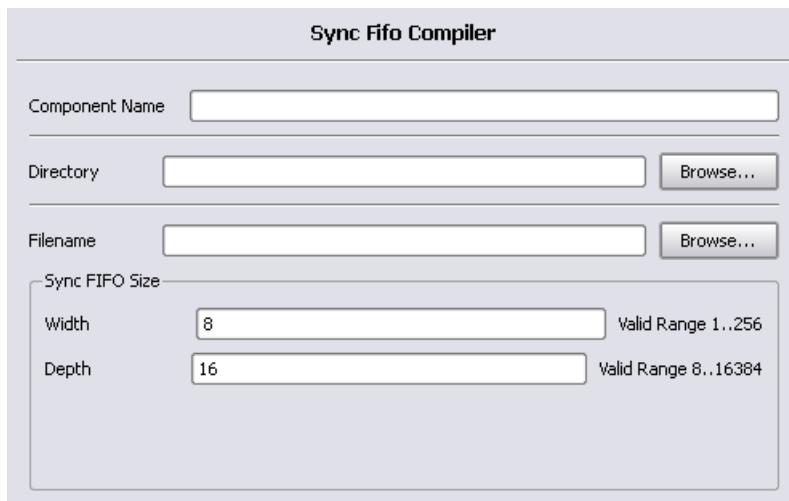
SYNCore FIFO Wizard

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 340](#)
- [SYNCore FIFO Parameters Page 2, on page 341](#)
- [SYNCore FIFO Parameters Page 3, on page 343](#)
- [SYNCore FIFO Parameters Page 4, on page 345](#)
- [SYNCore FIFO Parameters Page 5, on page 347](#)

SYNCore FIFO Parameters Page 1

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



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.

Parameter	Function
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.

SYNCORE FIFO Parameters Page 2



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	<p>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.</p> <p>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 355 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
Almost Full Flag	<p>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.</p> <p>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 355 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
Overflow Flag	<p>Specifies an Overflow signal, which is asserted to indicate that the write operation was unsuccessful because the FIFO was full.</p> <p>Enabling this option makes the Active High and Active Low options available for the signal (OVERFLOW_FLAG_SENSE parameter). See Handshaking Flags , on page 356 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
Write Acknowledge Flag	<p>Specifies a Write_ack signal, which is asserted at the completion of a successful write operation.</p> <p>Enabling this option makes the Active High and Active Low options (WACK_FLAG_SENSE parameter) available for the signal. See Handshaking Flags , on page 356 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

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	<p>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.</p> <p>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 356 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>

Parameter	Function
Almost Empty Flag	<p>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.</p> <p>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 356 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
Underflow Flag	<p>Specifies an Underflow signal, which is asserted to indicate that the read operation was unsuccessful because the FIFO was empty.</p> <p>Enabling this option makes the Active High and Active Low options (UNDRFLW_FLAG_SENSE parameter) available for the signal. See Handshaking Flags , on page 356 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
Read Acknowledge Flag	<p>Specifies a Read_ack signal, which is asserted at the completion of a successful read operation.</p> <p>Enabling this option makes the Active High and Active Low options (RACK_FLAG_SENSE parameter) available for the signal. See Handshaking Flags , on page 356 and FIFO Parameters , on page 353 for descriptions of the flag and parameter.</p>
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



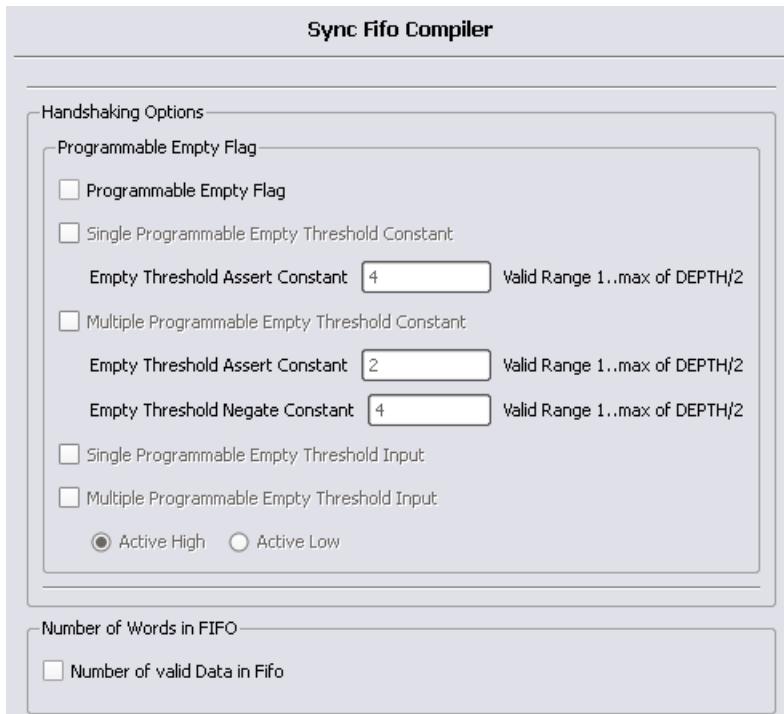
The page 4 parameters let you specify optional handshaking flags for FIFO programmable full operations. To use these options, you must have a Full signal specified. See [FIFO Programmable Flags, on page 358](#) for details and [FIFO Parameters, on page 353](#) for a list of the FIFO parameters. Data 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 359 and FIFO Parameters, on page 353 for descriptions of the flag and parameter.

Parameter	Function
Single Programmable Full Threshold Constant	<p>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 359 for details.</p> <p>Enabling this option makes Full Threshold Assert Constant available.</p>
Multiple Programmable Full Threshold Constant	<p>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 360 for details.</p> <p>Enabling this option makes Full Threshold Assert Constant and Full Threshold Negate Constant available.</p>
Full Threshold Assert Constant	<p>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.</p>
Full Threshold Negate Constant	<p>Specifies a constant that is used as a threshold value for de-asserting the Prog_full signal (PGM_FULL_NTHRESH parameter).</p>
Single Programmable Full Threshold Input	<p>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 360 for details.</p> <p>Enabling this option adds the Prog_full_thresh input port to the FIFO.</p>
Multiple Programmable Full Threshold Input	<p>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 Inputs , on page 360 for details.</p> <p>Enabling this option adds the Prog_full_thresh_assert and Prog_full_thresh_negate input ports to the FIFO.</p>
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

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 358](#) for details and [FIFO Parameters, on page 353](#) for a list of the FIFO parameters. Data is written/read on the rising edge of the clock.



Parameter	Function
Programmable Empty Flag	<p>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 361 and FIFO Parameters, on page 353 for descriptions of the flag and parameter.</p> <p>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.</p>

Parameter	Function
Single Programmable Empty Threshold Constant	<p>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 363 for details.</p> <p>Enabling this option makes Empty Threshold Assert Constant available.</p>
Multiple Programmable Empty Threshold Constant	<p>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 362 for details.</p> <p>Enabling this option makes Empty Threshold Assert Constant and Empty Threshold Negate Constant available.</p>
Empty Threshold Assert Constant	<p>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_ATRESH parameter for PGM_EMPTY_TYPE=2.</p>
Empty Threshold Negate Constant	<p>Specifies a constant that is used as a threshold value for de-asserting the Prog_empty signal (PGM_EMPTY_NTHRESH parameter).</p>
Single Programmable Empty Threshold Input	<p>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 363 for details.</p> <p>Enabling this option adds the Prog_full_thresh input port to the FIFO.</p>
Multiple Programmable Empty Threshold Input	<p>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 363 for details.</p> <p>Enabling this option adds the input ports to the FIFO.</p>
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

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.

FIFO Read and Write Operations

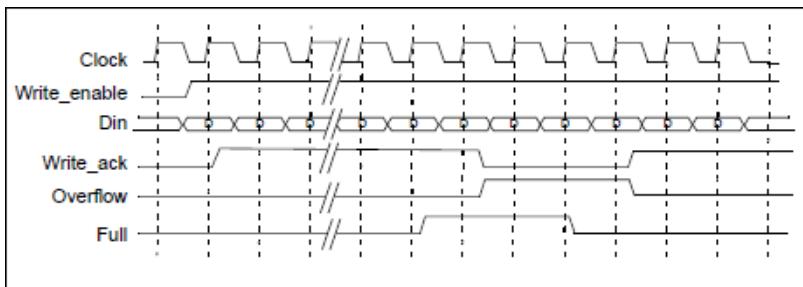
This section describes FIFO behavior with read and write operations.

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.

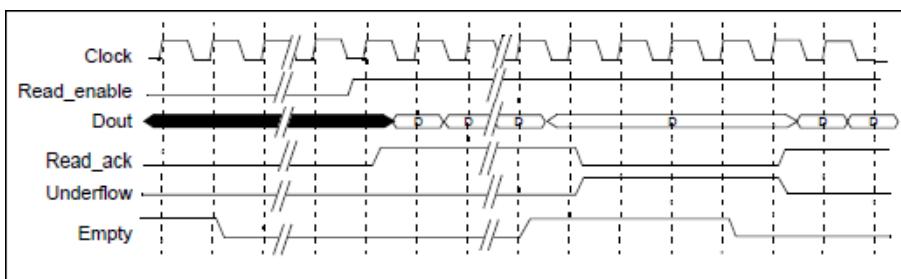


FIFO Read Operation

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 underflow of 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.



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.
Dout [width:0]	Data output word from the FIFO.

Port Name	Description
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.
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.

Port Name	Description
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 Low 1 Active High
OVERFLOW_FLAG_SENSE	FIFO overflow flag sense 0 Active Low 1 Active High

Parameter	Description
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 361 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 359 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
RACK_FLAG_SENSE	FIFO read acknowledge flag sense 0 Active Low 1 Active High

Parameter	Description
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.

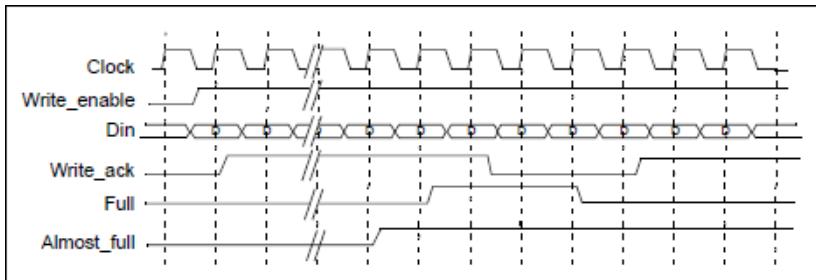
- [Full/Almost Full Flags, on page 355](#)
- [Empty/Almost Empty Flags, on page 356](#)
- [Handshaking Flags, on page 356](#)
- Programmable full and empty flags, which are described in [Programmable Full, on page 359](#) and [Programmable Empty, on page 361](#).

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.

The following figure displays the behavior of these flags. In this example, asserting Write_enable when Almost_full is high causes the assertion of Full on the next rising clock edge.

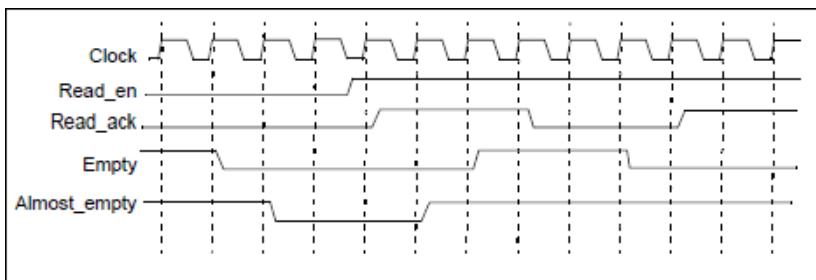


Empty/Almost Empty Flags

These flags indicate the status of the FIFO memory queue for read operations:

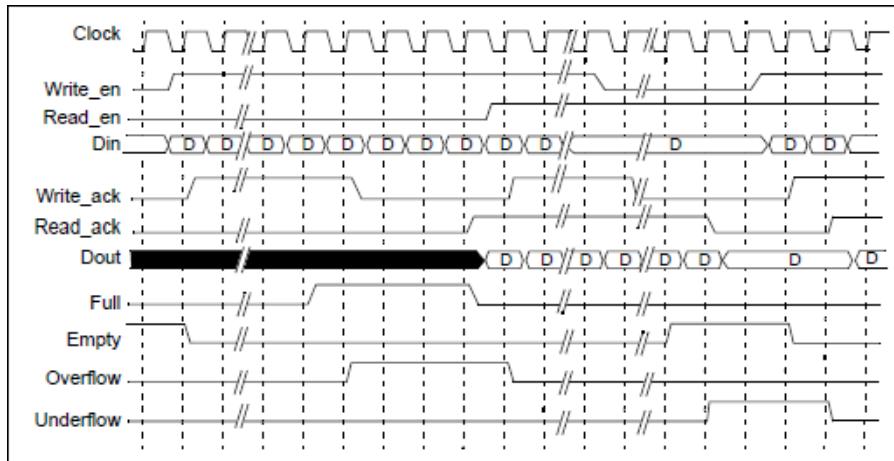
Empty	Indicates that the memory queue for the FIFO is empty and no more reads can be performed until data is written. The output is active high and is synchronous to the clock. If a read is initiated when the empty flag is true, the underflow flag is asserted.
Almost_empty	Indicates that the FIFO will 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.



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 359 for more information.
Prog_empty	Indicates that the FIFO has reached a user-defined empty threshold. See Programmable Empty , on page 361 for more information.

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

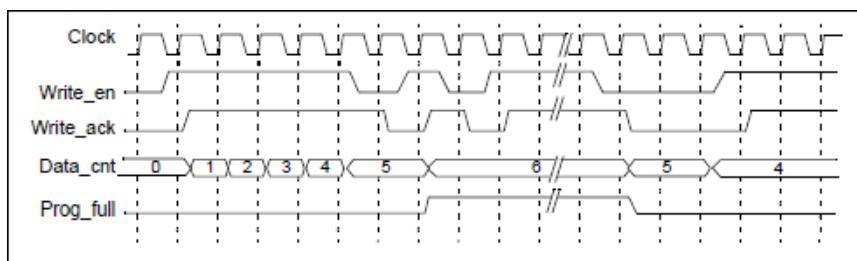
Prog_full has four threshold types:

- [Programmable Full with Single Threshold Constant, on page 359](#)
- [Programmable Full with Multiple Threshold Constants, on page 360](#)
- [Programmable Full with Single Threshold Input, on page 360](#)
- [Programmable Full with Multiple Threshold Inputs, on page 360](#)

Programmable Full with Single Threshold Constant

PGM_FULL_TYPE = 1

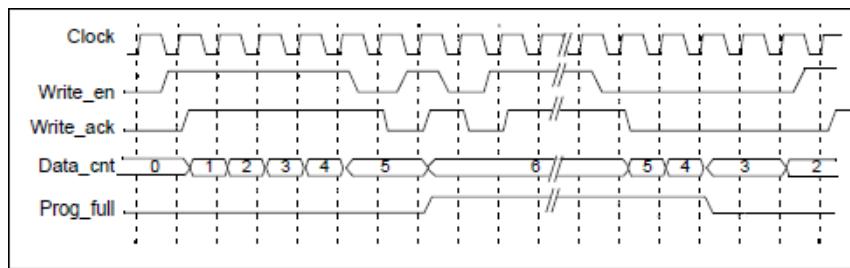
This option lets you set a single constant value for the threshold. It requires significantly fewer resources when the FIFO is generated. This figure illustrates the behavior of Prog_full when configured as a single threshold constant with a value of 6.



Programmable Full with Multiple Threshold Constants

PGM_FULL_TYPE = 2

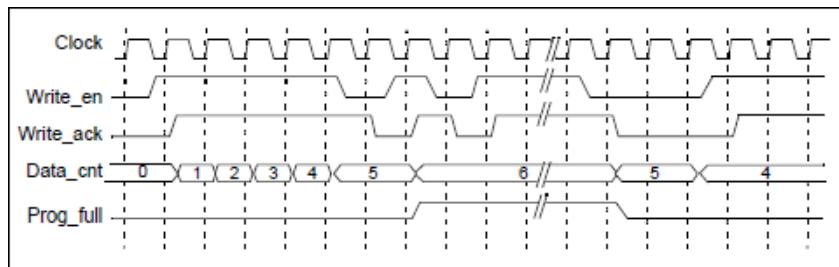
The programmable full flag is asserted when the number of words in the FIFO is greater than or equal to the full threshold assert value. If the number of FIFO words drops to less than the full threshold negate value, the programmable full flag is de-asserted. Note that the negate value must be set to 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.

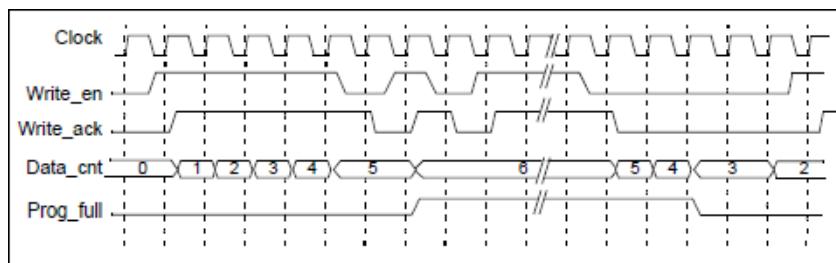


Programmable Full with Multiple Threshold Inputs

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.



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:

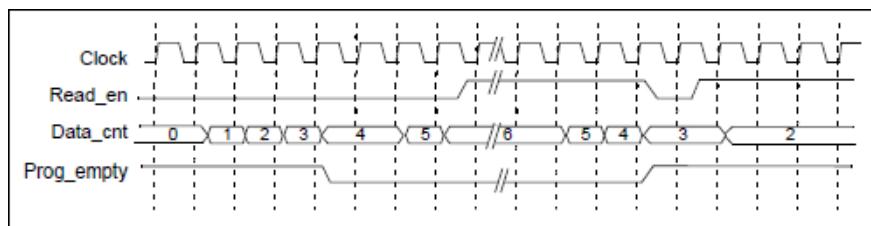
- [Programmable Empty with Single Threshold Constant](#), on page 362
- [Programmable Empty with Multiple Threshold Constants](#), on page 362
- [Programmable Empty with Single Threshold Input](#), on page 363

- [Programmable Empty with Multiple Threshold Inputs](#), on page 363

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.

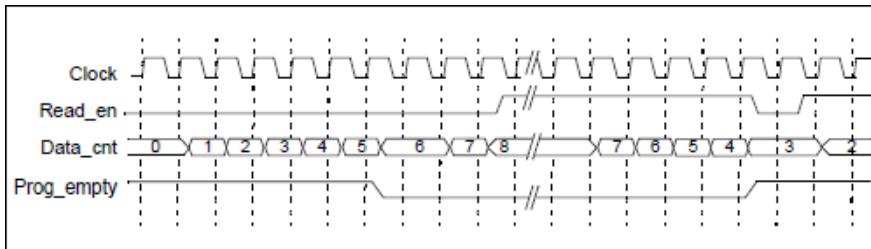


Programmable Empty with Multiple Threshold Constants

`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.

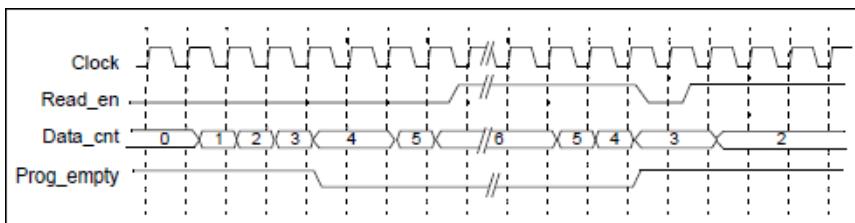
The following figure illustrates the behavior of `Prog_empty` when configured as multiple threshold constants with an assert value of 3 and a negate value of 5.



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.

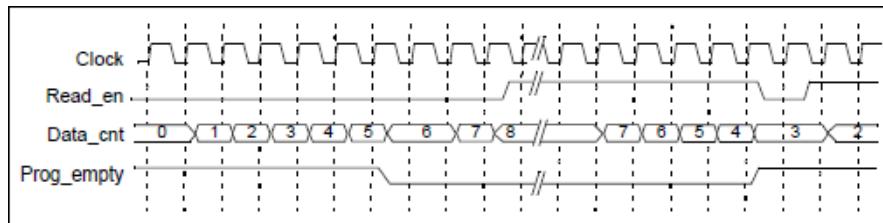


Programmable Empty with Multiple Threshold Inputs

`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.



SYNCore RAM Compiler

The SYNCore RAM Compiler generates Verilog code for your RAM implementation. This section describes the following:

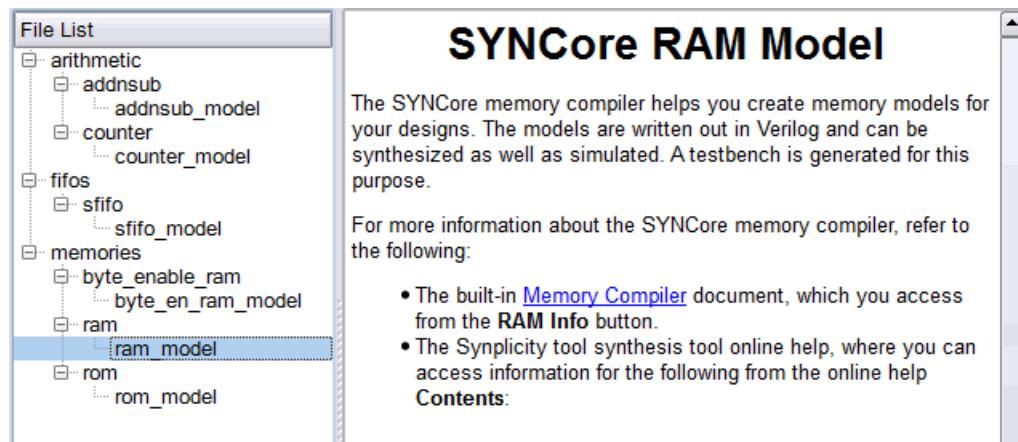
- [Specifying RAMs with SYNCore](#), on page 365
- [SYNCore RAM Wizard](#), on page 373
- [Single-Port Memories](#), on page 377
- [Dual-Port Memories](#), on page 379
- [Read/Write Timing Sequences](#), on page 383

Specifying RAMs with SYNCore

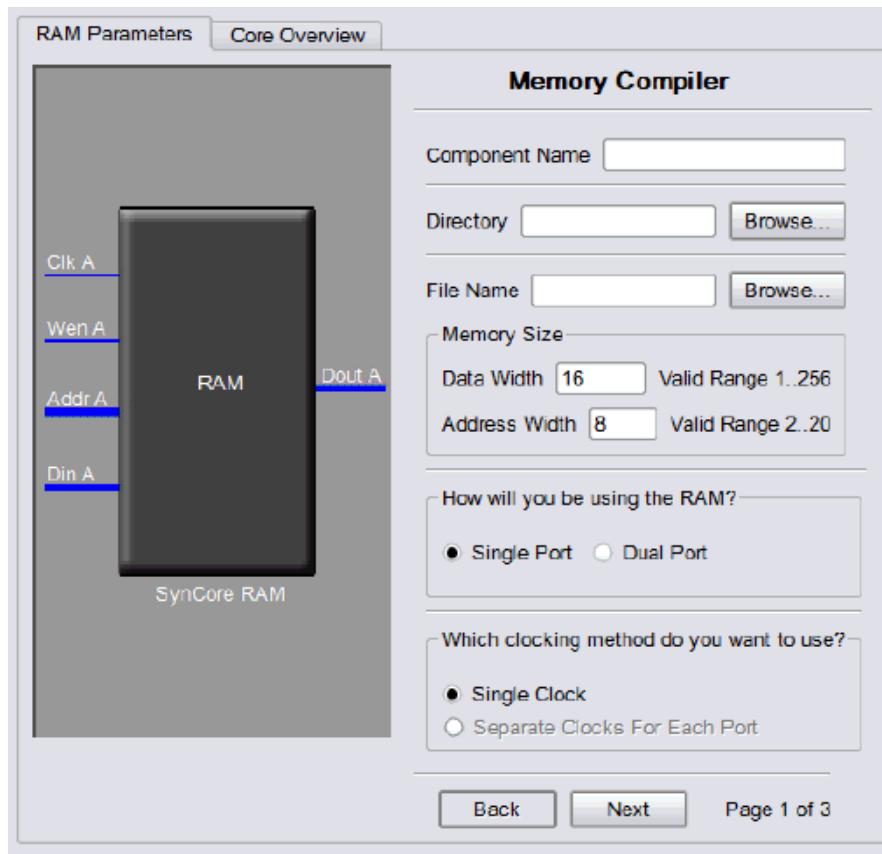
The SYNCore IP wizard helps you generate Verilog code for your RAM implementation requirements. The following procedure shows you how to generate Verilog code for a RAM using the SYNCore IP wizard.

Note: The SYNCore RAM model uses Verilog 2001. When adding a RAM model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a `set_option -vlog_std v2001` statement in your project file to prevent a syntax error.

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



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



2. Specify the parameters you need in the wizard.

- For details about the parameters for a single-port RAM, see [Specifying Parameters for Single-Port RAM, on page 370](#).
- For details about the parameters for a dual-port RAM, see [Specifying Parameters for Dual-Port RAM, on page 371](#). Note that dual-port implementations are only supported for some technologies.

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

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

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

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

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

You can now close the SYNCore Memory Compiler.

4. Edit the RAM files if necessary.

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

5. Add the RAM you generated to your design.

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

```
module top (
    input ClkA,
    input [7:0] AddrA,
    input [15:0] DataInA,
    input WrEnA,
    output [15:0] DataOutA
);

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

endmodule
```

template

6. Edit the template port connections so that they agree with the port definitions in your top-level module as shown in the example below (the updated connection names are shown in red). You can also assign a unique name to each instantiation.

```
module top (

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

    output [15:0] DataOutA

);

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

endmodule
```

Specifying Parameters for Single-Port RAM

To create a single-port RAM with the SYNCore Memory Compiler, you need to specify a single read/write address (single port) and a single clock. You only need to configure Port A. The following procedure lists what you need to specify. For descriptions of each parameter, refer to [SYNCore RAM Wizard, on page 373](#).

1. Start the SYNCore RAM wizard, as described in [Specifying RAMs with SYNCore, on page 365](#).
2. Do the following on page 1 of the RAM wizard:
 - In Component Name, specify a name for the memory. Do not use spaces.
 - In Directory, specify a directory where you want the output files to be written. Do not use spaces.
 - In Filename, specify a name for the Verilog file that will be generated with the RAM specifications. Do not use spaces.

- Enter data and address widths.
- Enable Single Port, to specify that you want to generate a single-port RAM. This automatically enables Single Clock.
- Click Next. The wizard opens another page where you can set parameters for Port A.

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

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

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

You can now generate the RAM by clicking Generate, as described in [Specifying RAMs with SYNCORE, on page 365](#). You do not need to specify any parameters on page 3, as this is a single-port RAM and you do not need to specify Port B. All output files are in the directory you specified on the first page of the wizard.

For details about setting dual-port RAM parameters, see [Specifying Parameters for Dual-Port RAM, on page 371](#). For read/write timing diagrams, see [Read/Write Timing Sequences, on page 383](#).

Specifying Parameters for Dual-Port RAM

The following procedure shows you how to set parameters for dual-port memory in the SYNCORE wizard. Dual-port RAMs are only supported for some technologies. For information about generating single-port RAMs, see [Specifying Parameters for Single-Port RAM, on page 370](#). It shows you how to generate these common RAM configurations:

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

For the corresponding read/write timing diagrams, see [Read/Write Timing Sequences, on page 383](#).

1. Start the SYNCore RAM wizard, as described in [Specifying RAMs with SYNCore, on page 365](#).
2. Do the following on page 1 of the RAM wizard:
 - In Component Name, specify a name for the memory. Do not use spaces.
 - In Directory, specify a directory where you want the output files to be written. Do not use spaces.
 - In Filename, specify a name for the Verilog file that will be generated with the RAM specifications. Do not use spaces.
 - Enter data and address widths.
 - Enable Dual Port, to specify that you want to generate a dual-port RAM.
 - Specify the clocks.

For a single clock ... Enable Single Clock.

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

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

One read & one write Enable Read Only Access.

Two reads & one write Enable Read and Write Access.
Specify a setting for Use Write Enable.

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

- Specify a setting for Register Read Address.
- Set Synchronous Reset to the setting you want. Register Outputs is always enabled.

- Click Next. The wizard opens another page where you can set parameters for Port B. The page and the parameters are identical to the previous page, except that the settings are for Port B instead of Port A.
4. Specify the settings for Port B on page 3 of the wizard according to the kind of memory you want to generate:

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

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

You can now generate the RAM by clicking Generate, as described in [Specifying RAMs with SYNCORE](#), on page 365, and add it to your design.

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 374
- [SYNCORE RAM Parameters Pages 2 and 3](#), on page 376

SYNCore RAM Parameters Page 1

Memory Compiler

Component Name

Directory

Filename

Memory Size

Data Width Valid Range 1..256

Address Width Valid Range 2..256

How will you be using the RAM?

Single Port Dual Port

Which clocking method do you want to use?

Single Clock Separate Clocks For Each Port

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:

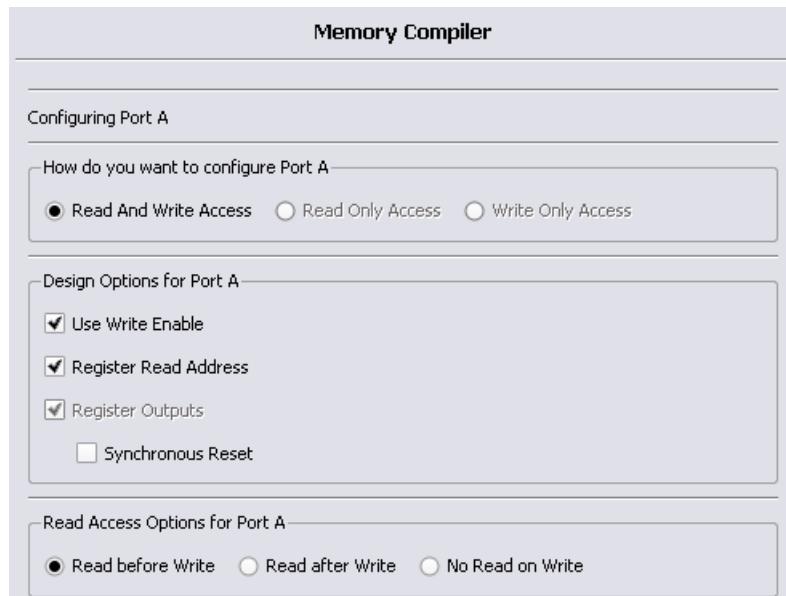
```
ram101 <ComponentName> (
    .PortAClk(PortAClk)
    , .PortAAddr(PortAAddr)
    , .PortADataIn(PortADataIn)
    , .PortAWriteEnable(PortAWriteEnable)
    , .PortBDataIn(PortBDataIn)
    , .PortBAddr(PortBAddr)
    , .PortBWriteEnable(PortBWriteEnable)
    , .PortADataOut(PortADataOut)
    , .PortBDataOut(PortBDataOut)
);
```

Directory	Specifies the directory where the generated files are stored. Do not use spaces. The following files are created: <ul style="list-style-type: none"> • 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.



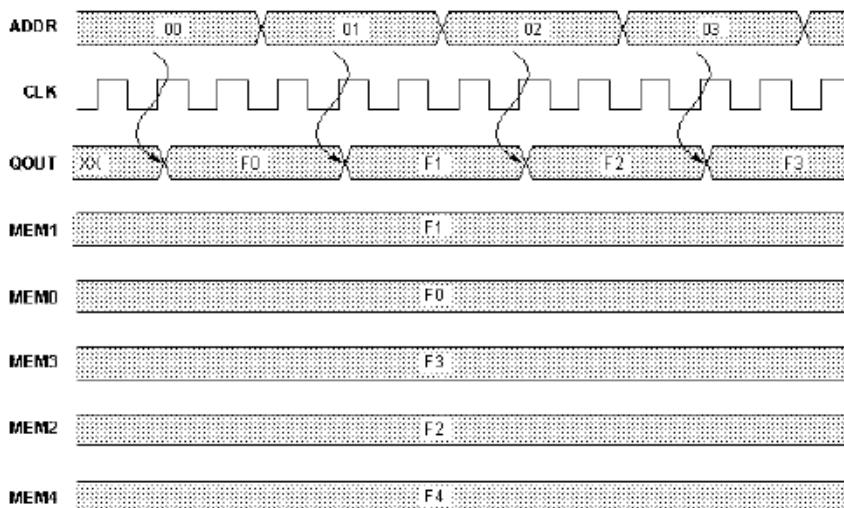
Read and Write Access	Specifies that the port can be accessed by both read and write operations.
Read Only Access	Specifies that the port can only be accessed by read operations.
Write Only Access	Specifies that the port can only be accessed by write operations.
Use Write Enable	Includes write-enable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

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 384 .
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 385 .
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 386 .

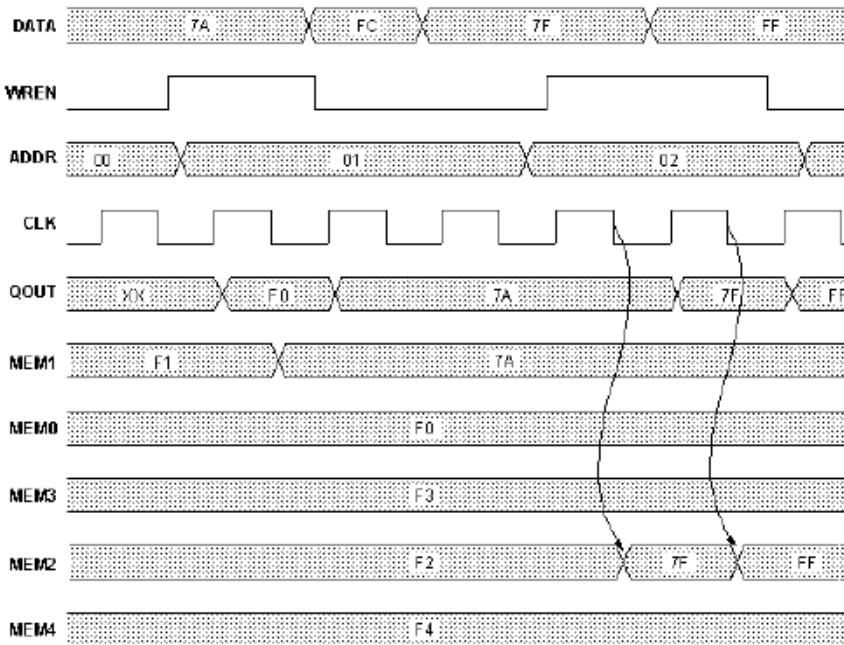
Single-Port Memories

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 365](#) for a procedure.

Single-Port Read



Single-Port Write



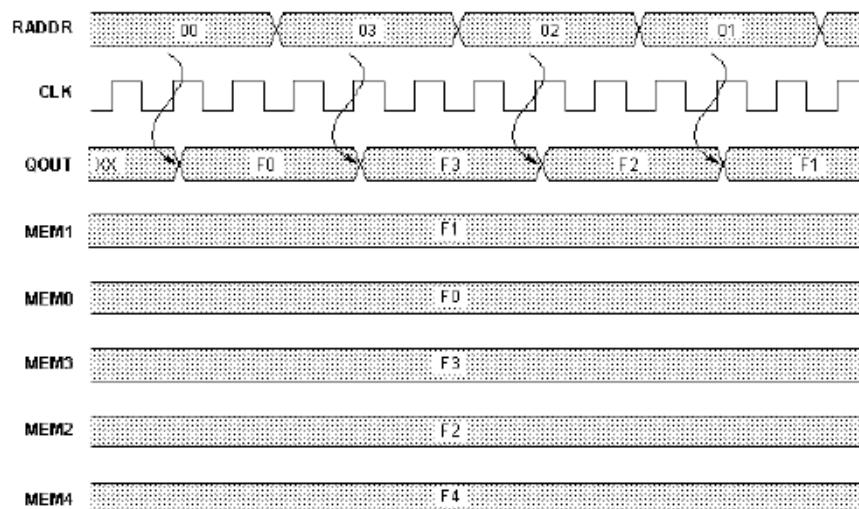
Dual-Port Memories

SYNCore dual-port memory includes the following common configurations:

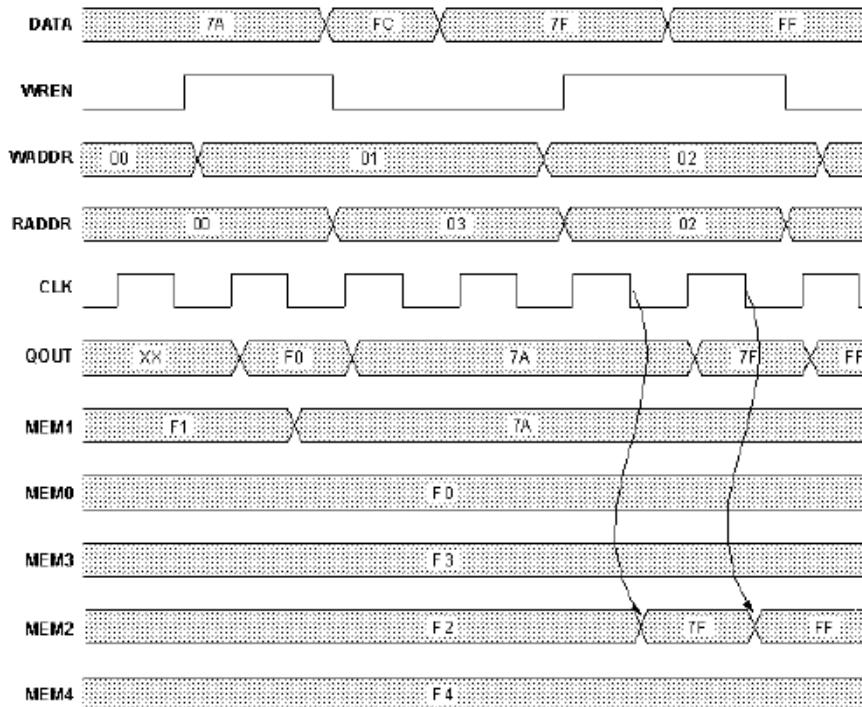
- One read access and one write access
- Two read accesses and one write access
- Two read accesses and two write accesses

The following diagrams show the read-write timing for dual-port memories. See [Specifying RAMs with SYNCore](#), on page 365 for a procedure to specify a dual-port RAM with SYNCore.

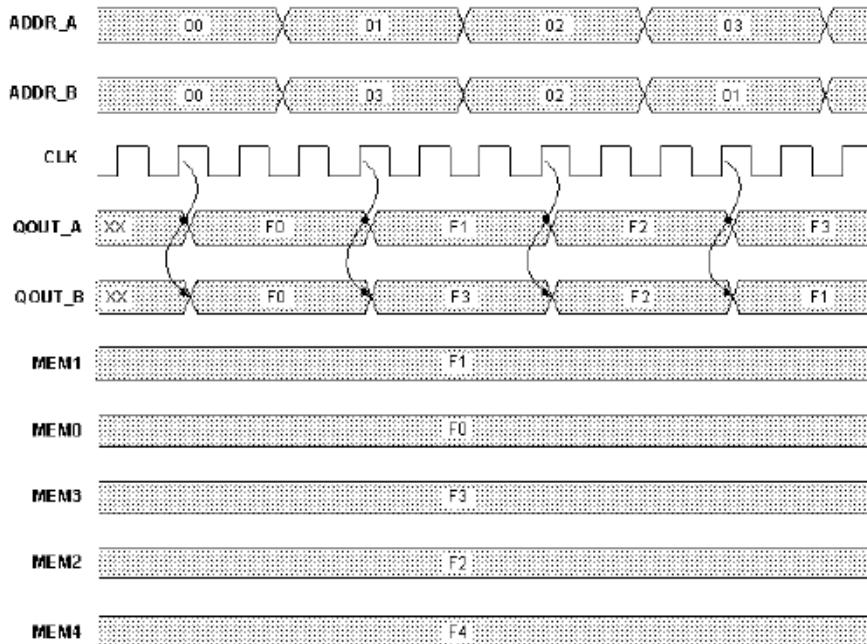
Dual-Port Single Read



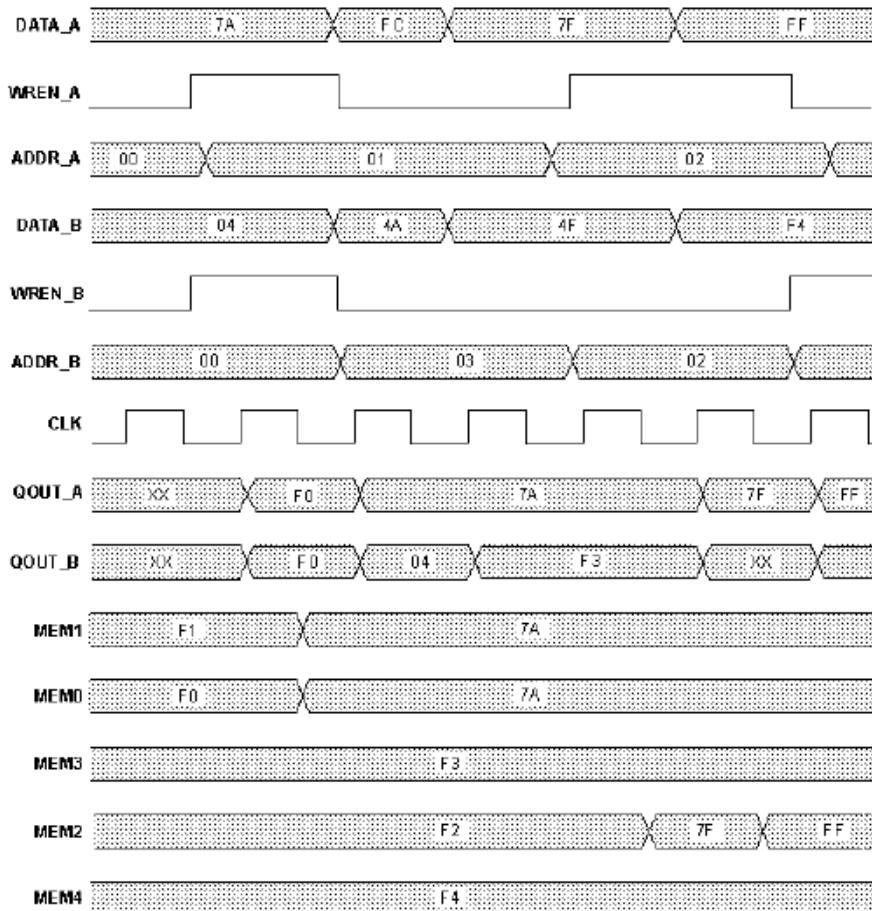
Dual-Port Single Write



Dual-Port Read



Dual-Port Write



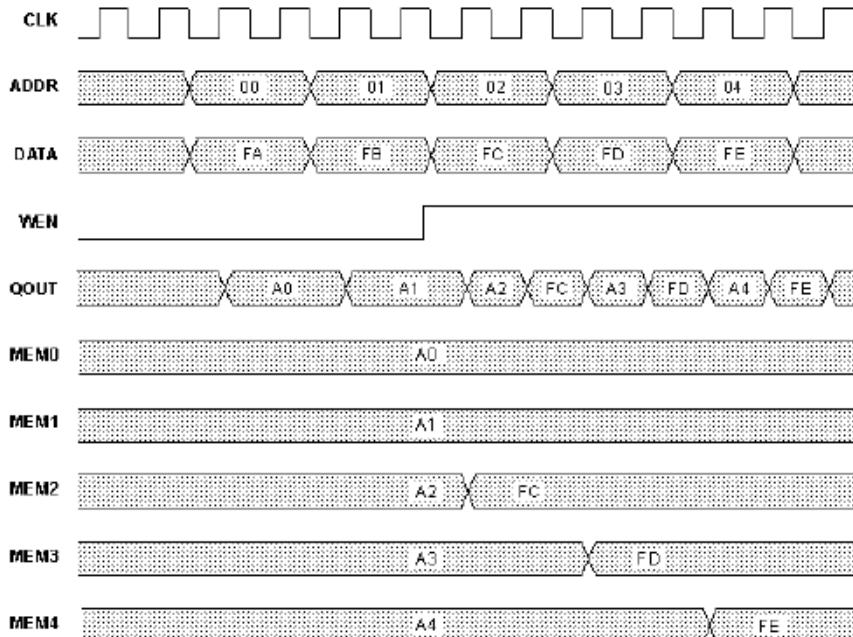
Read/Write Timing Sequences

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:

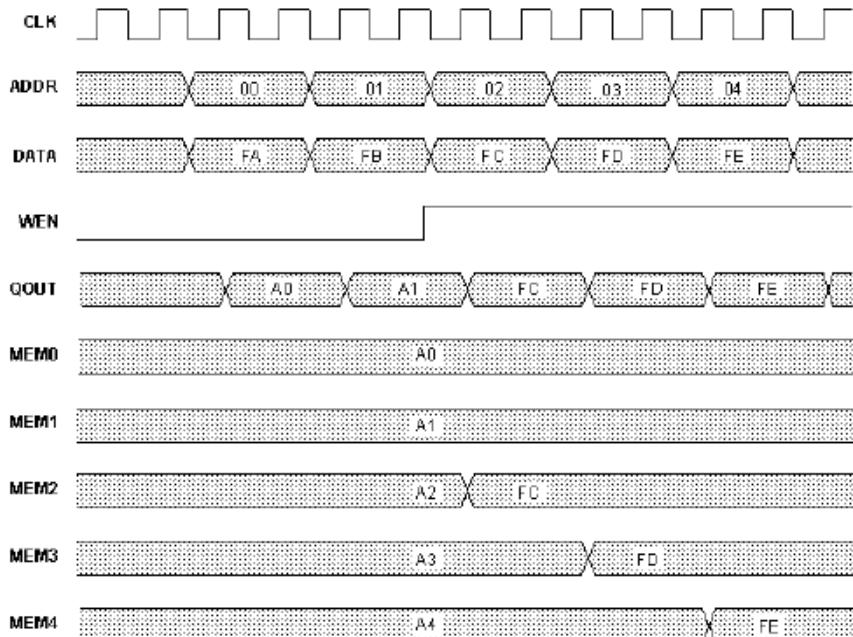
- [Read Before Write](#), on page 384

- [Write Before Read](#), on page 385
- [No Read on Write](#), on page 386

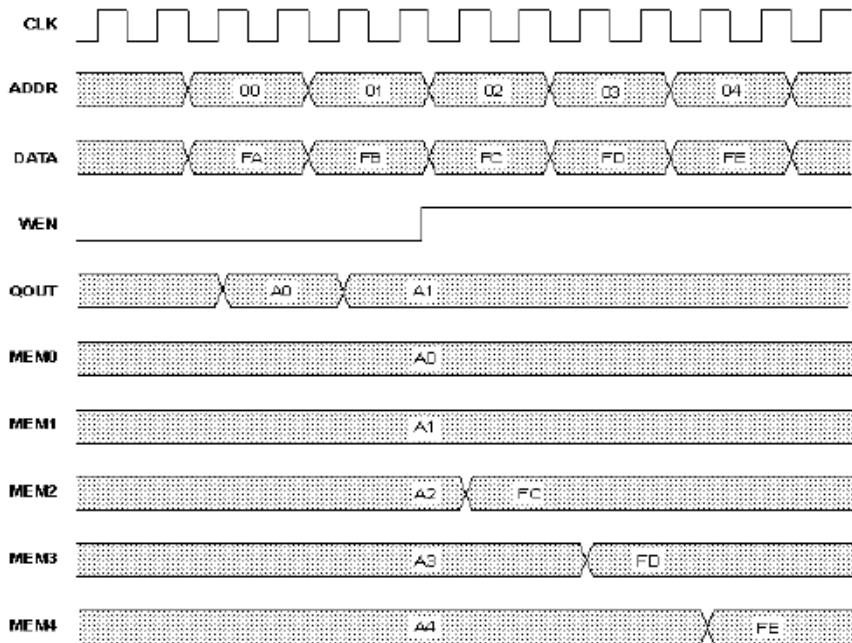
Read Before Write



Write Before Read



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 387
- [Specifying Byte-Enable RAMs with SYNCORE](#), on page 388
- [SYNCORE Byte-Enable RAM Wizard](#), on page 395
- [Read/Write Timing Sequences](#), on page 398
- [Parameter List](#), on page 401

Functional Overview

The SYNCORE byte-enable RAM component supports bit/byte-enable RAM implementations using block RAM 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.

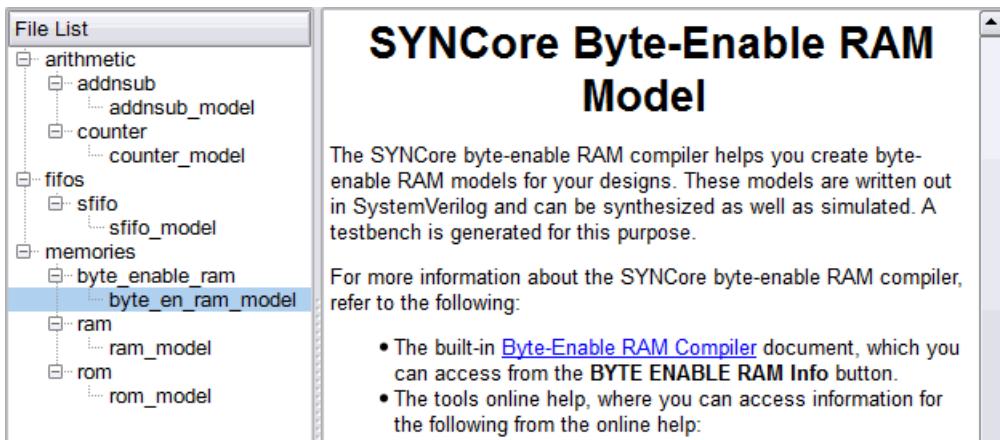
Specifying Byte-Enable RAMs with SYNCore

The SYNCore IP wizard helps you generate SystemVerilog code for your byte-enable RAM implementation requirements. The following procedure shows you how to generate SystemVerilog code for a byte-enable RAM using the SYNCore IP wizard.

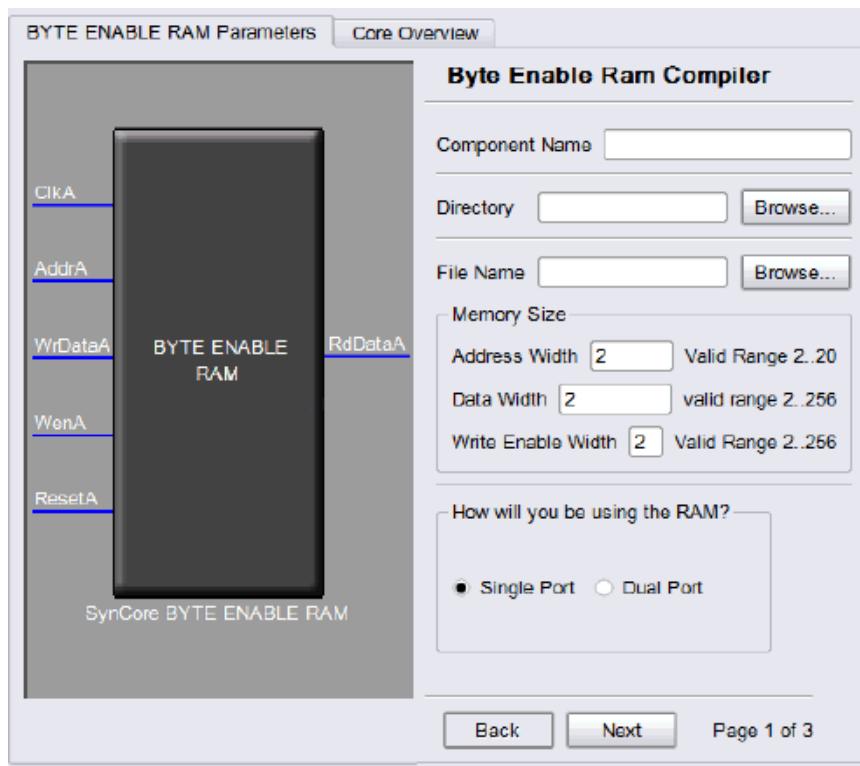
Note: The SYNCore byte-enable RAM model uses SystemVerilog. When adding a byte-enable RAM to your design, be sure to enable the System Verilog check box on the Verilog tab of the Implementation Options dialog box or include a `set_option -vlog_std sysv` statement in your project file to prevent a syntax error.

1. Start the wizard.

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



- In the window that opens, select `byte_en_ram_model` and click Ok to open the first page (page1) of the wizard.



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying Byte-Enable RAM Parameters, on page 392](#). The BYTE ENABLE RAM symbol on the left reflects any parameters you set.
3. After you have specified all the parameters you need, click the Generate button in the lower left corner. The tool displays a confirmation message (TCL execution successful!) and writes the required files to the directory you specified on page 1 of the wizard. The HDL code is in SystemVerilog.
SYNCORE also generates a test bench for the byte-enable RAM component. The test bench covers a limited set of vectors. You can now close the SYNCORE byte-enable RAM compiler.
4. Edit the generated files for the byte-enable RAM component if necessary.
5. Add the byte-enable RAM that you generated to your design.

- On the Verilog tab of the Implementation Options dialog box, make sure that SystemVerilog is enabled.
- Use the Add File command to add the Verilog design file that was generated (the filename entered on page 1 of the wizard) and the `syncore_*.v` file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
- Use a text editor to open the `instantiation_file.vin` template file. This file is located in the same output files directory. Copy the lines that define the byte-enable RAM and paste them into your top-level module.
- Edit the template port connections so that they agree with the port definitions in the top-level module; also change the instantiation name to agree with the component name entered on page 1. The following figure shows a template file inserted into a top-level module with the updated component name and port connections in red.

```

module top
  (input ClockA,
   input [3:0] AddA
   input [31:0] DataIn
   input WrEnA,
   input Reset
   output [31:0] DataOut
  )|  

INST_TAG  

SP_RAM #  

  (.ADD_WIDTH(4),
   .WE_WIDTH(2),
   .RADDR_LTNCY_A(1), // 0 - No Latency, 1 - 1 Cycle Latency
   .RDATA_LTNCY_A(1), // 0 - No Latency, 1 - 1 Cycle Latency
   .RST_TYPE_A(1), // 0 - No Reset, 1 synchronous
   .RST_RDATA_A({32{1'b1}}),
   .DATA_WIDTH(32)
  )  

4x32spram
  ( // Output Ports
   .RdDataA(DataIn),
   // Input Ports
   .WrDataA(DataOut),
   .WenA(WeEnA),
   .AddrA(AddA),
   .ResetA(Reset),
   .ClkA(ClockA)
  ;

```

Port List

Port A interface signals are applicable for both single-port and dual-port configurations; Port B signals are applicable for dual-port configuration only.

Name	Type	Description
ClkA	Input	Clock input for Port A
WenA	Input	Write enable for Port A; present when Port A is in write mode
AddrA	Input	Memory access address for Port A

ResetA	Input	Reset for memory and all registers in core; present with registered read data when Reset is enabled; active low (cannot be changed)
WrDataA	Input	Write data to memory for Port A; present when Port A is in write mode
RdDataA	Output	Read data output for Port A; present when Port A is in read or read/write mode
ClkB	Input	Clock input for Port B; present in dual-port mode
WenB	Input	Write enable for Port B; present in dual-port mode when Port B is in write mode
AddrB	Input	Memory access address for Port B; present in dual-port mode
ResetB	Input	Reset for memory and all registers in core; present in dual-port mode when read data is registered and Reset is enabled; active low (cannot be changed)
WrDataB	Input	Write data to memory for Port B; present in dual-port mode when Port B is in write mode
RdDataB	Output	Read data output for Port B; present in dual-port mode when Port B is in read or read/write mode

Specifying Byte-Enable RAM Parameters

When creating a single-port, byte-enable RAM with the SYNCORE IP wizard, you must specify a single read address and a single clock; you only need to configure the Port A parameters on page 2 of the wizard.

When creating a dual-port, byte-enable RAM, you must additionally configure the Port B parameters on page 3 of the wizard.

The following procedure lists the parameters you need to specify. For descriptions of each parameter, refer to [Parameter List, on page 401](#).

1. Start the SYNCORE byte-enable RAM wizard as described in [Specifying Byte-Enable RAMs with SYNCORE, on page 388](#).

2. Do the following on page 1 of the byte-enable RAM wizard:

- Specify a name for the memory in the Component Name field; do not use spaces.
- Specify a directory name in the Directory field where you want the output files to be written; do not use spaces.
- Specify a name in the File Name field for the SystemVerilog file to be generated with the byte-enable RAM specifications; do not use spaces.
- Enter a value for the address width of the byte-enable RAM; the maximum depth of memory is limited to 2^{256} .
- Enter a value for the data width for the byte-enable RAM; data width values range from 2 to 256.
- Enter a value for the write enable width; write-enable width values range from 1 to 4.
- Select Single Port to generate a single-port, byte-enable RAM or select Dual Port to generate a dual-port, byte-enable RAM.
- Click Next to open page 2 of the wizard.

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

3. Do the following on page 2 (configuring Port A) of the wizard:

- Select the Port A configuration. Only Read and Write Access mode is valid for single-port configurations; this mode is selected by default.
- Set Pipelining Address Bus and Output Data according to your application. By default, read data is registered; you can register both the address and data registers.
- Set the Configure Reset Options. Enabling the checkbox enables the synchronous reset for read data. This option is enabled only when the read data is registered. Reset is active low and cannot be changed.
- Configure output reset data value options under Specify output data on reset; reset data can be set to default value of all '1' s or to a user-defined decimal value. Reset data value options are disabled when the reset is not enabled for Port A.
- Set Write Enable for Port A value; default for the write-enable level is active high.

4. If you are generating a dual-port, byte-enable RAM, set the Port B parameters on page 3 (note that the Port B parameters are only enabled when Dual Port is selected on page 1).

The Port B parameters are identical to the Port A parameters on page 2. When using the dual-port configuration, when one port is configured for read access, the other port can only be configured for read/write access or write access.

5. Generate the byte-enable RAM by clicking Generate. Add the file to your project and edit the template file as described in *Specifying Byte-Enable RAMs with SYNCore*, on page 388. For read/write timing diagrams, see *Read/Write Timing Sequences*, on page 383.

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 395](#)
- [SYNCore Byte-Enable RAM Parameters Pages 2 and 3, on page 396](#)

SYNCore Byte-Enable RAM Parameters Page 1

Byte Enable Ram Compiler

Component Name

Directory

File Name

Memory Size

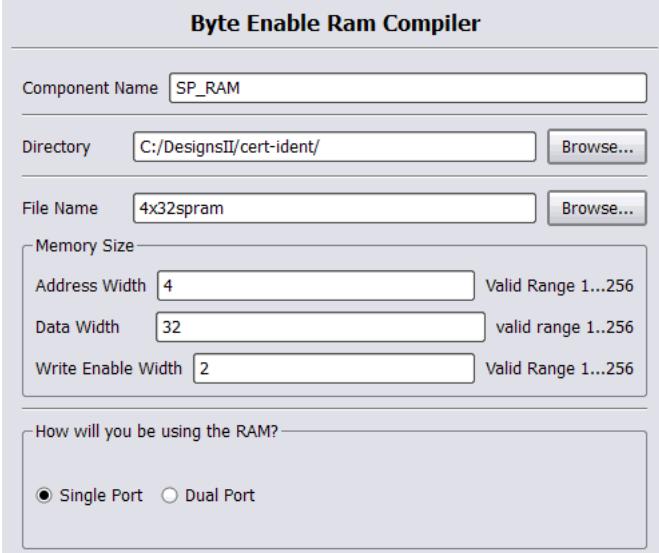
Address Width Valid Range 1...256

Data Width valid range 1..256

Write Enable Width Valid Range 1...256

How will you be using the RAM?

Single Port Dual Port



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	<p>Specifies the directory where the generated files are stored. Do not use spaces. The following files are created:</p> <ul style="list-style-type: none"> • <code>filelist.txt</code> - lists files written out by SYNCore • <code>options.txt</code> - lists the options selected in SYNCore • <code>readme.txt</code> - contains a brief description and known issues • <code>syncore_be_ram_sdp.v</code> - SystemVerilog library file required to generate single or simple dual-port, byte-enable RAM model • <code>syncore_be_ram_tdp.v</code> - SystemVerilog library file required to generate true dual-port byte-enable RAM model • <code>testbench.v</code> - Verilog testbench file for testing the byte-enable RAM model • <code>instantiation_file.vin</code> - describes how to instantiate the wrapper file • <code>component.v</code> - Byte-enable RAM model wrapper file generated by SYNCore <p>Note that running the byte-enable RAM wizard in the same directory overwrites the existing files.</p>
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.

Byte Enable Ram Compiler

Configuring Port A

How do you want to configure Port A

- Read And Write Access Read Only Access Write Only Access

Pipelining Address Bus and Output Data

- Register address bus AddrA
 Register output data bus RdDataA

Configure Reset Options

- Reset for RdDataA

Specify output data on reset

- Default value of '1' for all bits
 Specify Reset value for RdDataA Valid Range 0... $2^{\text{DATA_WIDTH}}$

Configure Write Enable Options

- Write Enable for PORTA
 Active High Active Low

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.

Reset for RdDataA/B	Specifies the reset type for registered read data: <ul style="list-style-type: none">• 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): <ul style="list-style-type: none">• 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.

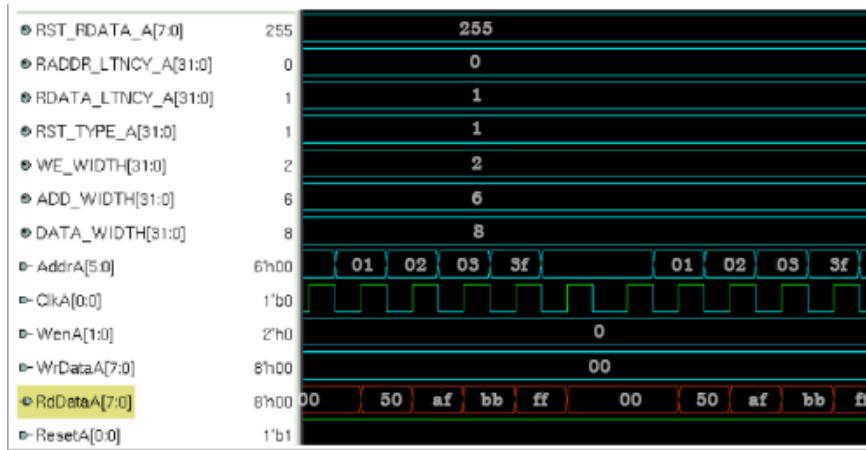
Read/Write Timing Sequences

The waveforms in this section describe the behavior of the byte-enable RAM for both read and write operations.

Read Operation

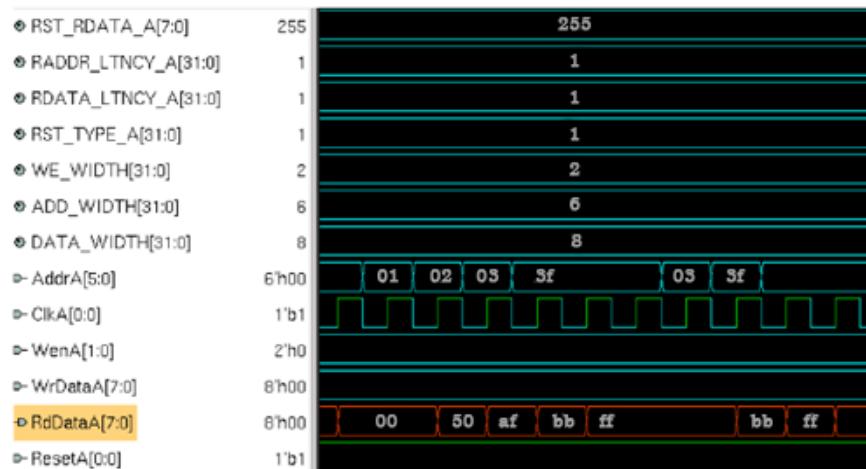
On each active edge of the clock when there is a change in address, data is valid on the same clock or next clock (depending on latency parameter values for read address and read data ports). Active reset ignores any change in input address, and data and output data are initialized to user-defined values 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.

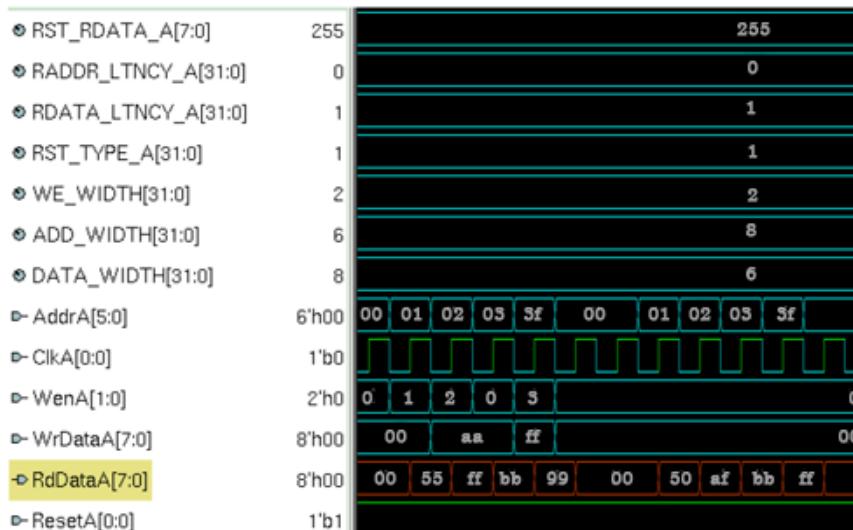


As shown in the above waveform, output read data changes on the next clock edge after the input address changes. When the address changes from 'h00 to 'h01, read data changes to 50 on the next clock, and data is valid on the next clock edge.

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.



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.

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	

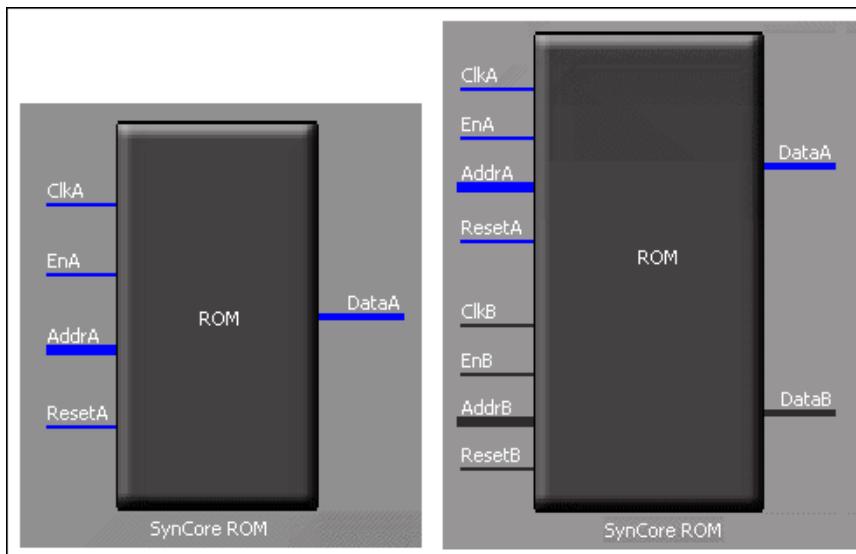
SYNCore ROM Compiler

The SYNCore ROM Compiler generates Verilog code for your ROM implementation. This section describes the following:

- [Functional Overview](#), on page 403
- [Specifying ROMs with SYNCore](#), on page 405
- [SYNCore ROM Wizard](#), on page 410
- [Single-Port Read Operation](#), on page 414
- [Dual-Port Read Operation](#), on page 415
- [Parameter List](#), on page 416
- [Clock Latency](#), on page 417

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.



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 417](#)). 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.

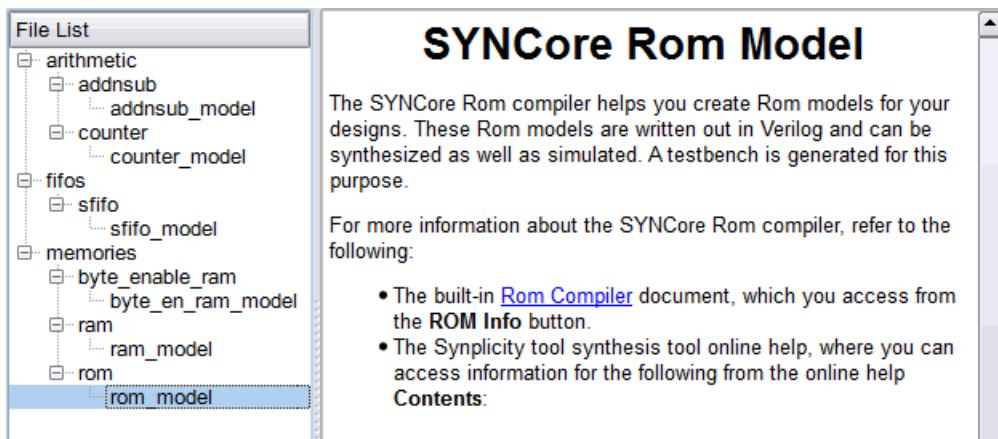
Specifying ROMs with SYNCore

The SYNCore IP wizard helps you generate Verilog code for your ROM implementation requirements. The following procedure shows you how to generate Verilog code for a ROM using the SYNCore IP wizard.

Note: The SYNCore ROM model uses Verilog 2001. When adding a ROM model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a `set_option -vlog_std v2001` statement in your project file to prevent a syntax error.

1. Start the wizard.

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



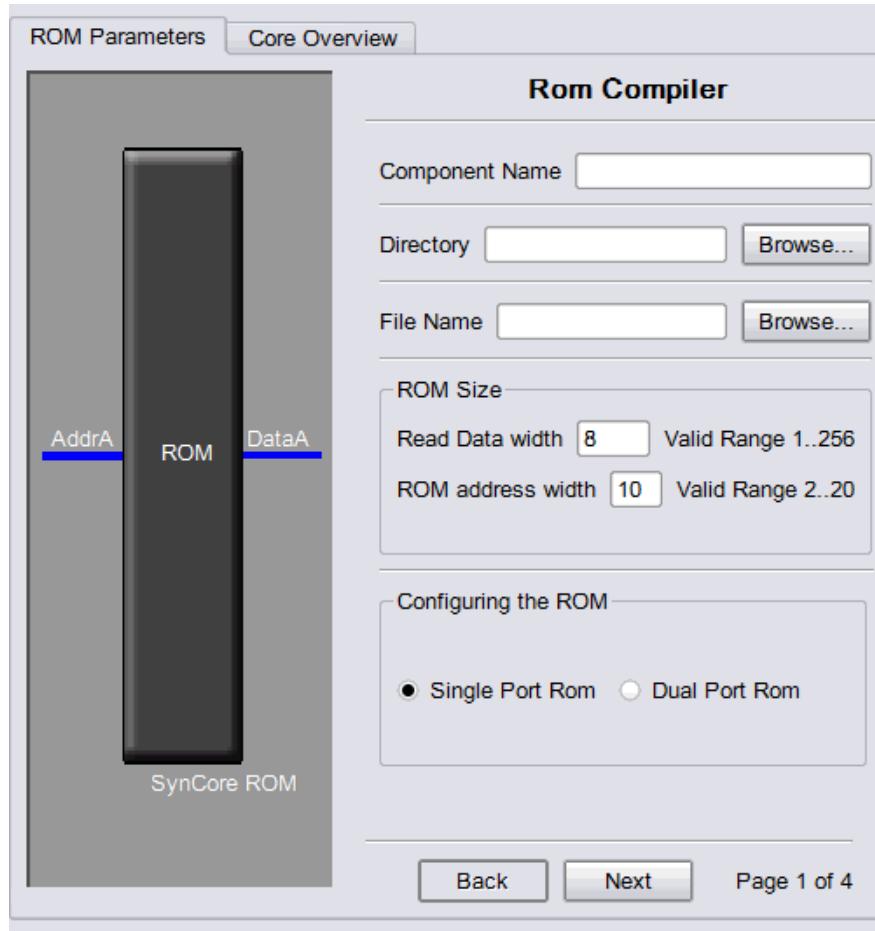
SYNCore Rom Model

The SYNCore Rom compiler helps you create Rom models for your designs. These Rom models are written out in Verilog and can be synthesized as well as simulated. A testbench is generated for this purpose.

For more information about the SYNCore Rom compiler, refer to the following:

- The built-in [Rom Compiler](#) document, which you access from the **ROM Info** button.
- The Synplicity tool synthesis tool online help, where you can access information for the following from the online help **Contents**:

- In the window that opens, select `rom_model` and click Ok to open page 1 of the wizard.



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying ROM Parameters, on page 409](#). The ROM symbol on the left reflects any parameters you set.
3. After you have specified all the parameters you need, click the Generate button in the lower left corner. The tool displays a confirmation message (TCL execution successful!) and writes the required files to the directory you specified on page 1 of the wizard. The HDL code is in Verilog.

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

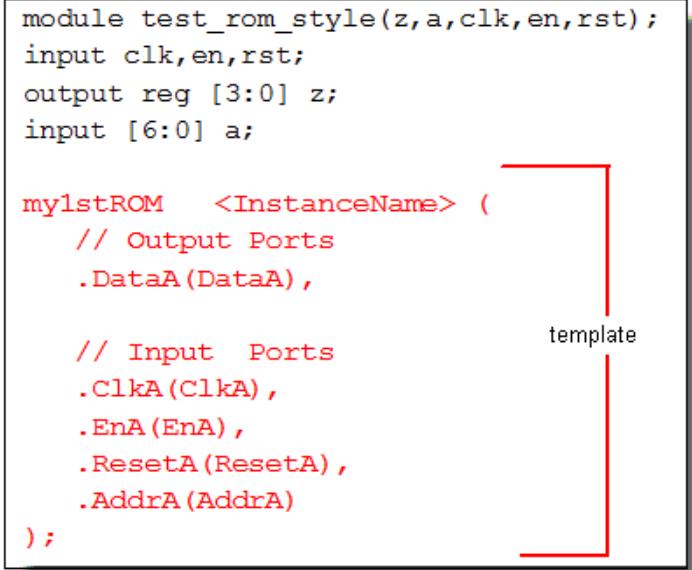
You can now close the SYNCORE ROM Compiler.

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

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

  my1stROM <InstanceName> (
    // Output Ports
    .DataA(DataA),
    // Input Ports
    .ClkA(ClkA),
    .EnA(EnA),
    .ResetA(ResetA),
    .AddrA(AddrA)
  );

```



A red bracket is drawn from the text "template" to the instantiation code within the module definition, specifically pointing to the line "my1stROM <InstanceName> (".

6. Edit the template port connections so that they agree with the port definitions in your top-level module as shown in the example below (the updated connection names are shown in red). You can also assign a unique name to each instantiation.

```

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

my1stROM decode_rom(
  // Output Ports
  .DataA(z),
  // Input Ports
  .ClkA(clk),
  .EnA(en),
  .ResetA(rst),
  .AddrA(a)
);

```

Port List

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

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

Specifying ROM Parameters

If you are creating a single-port ROM with the SYNCore IP wizard, you need to specify a single read address and a single clock, and you only need to configure the Port A parameters on page 2. If you are creating a dual-port ROM, you must additionally configure the Port B parameters on page 3. The following procedure lists what you need to specify. For descriptions of each parameter, refer to [SYNCore RAM Wizard, on page 373](#).

1. Start the SYNCore ROM wizard, as described in [Specifying ROMs with SYNCore, on page 405](#).
2. Do the following on page 1 of the ROM wizard:
 - In Component Name, specify a name for the memory. Do not use spaces.
 - In Directory, specify a directory where you want the output files to be written. Do not use spaces.
 - In Filename, specify a name for the Verilog file that will be generated with the ROM specifications. Do not use spaces.
 - Enter values for Read Data width and ROM address width (minimum depth value is 2; maximum depth of the memory is limited to 2^{256}).
 - Select Single Port Rom to indicate that you want to generate a single-port ROM or select Dual Port Rom to generate a dual-port ROM.
 - Click Next. The wizard opens page 2 where you set parameters for Port A.

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

3. Do the following on page 2 (Configuring Port A) of the RAM wizard:
 - For synchronous ROMs, select Register address bus AddrA and/or Register output data bus DataA to register the read address and/or the outputs. Selecting either checkbox enables the Enable for Port A checkbox which is used to select the Enable level.
 - Set the Configure Reset Options. Enabling the checkbox enables the type of reset (asynchronous or synchronous) and allows an output data pattern (all 1's or a specified pattern) to be defined on page 4.
4. If you are generating a dual-port ROM, set the port B parameters on page 3 (the page 3 parameters are only enabled when Dual Port Rom is selected on page 1).

5. On page 4, specify the location of the ROM initialization file and the data format (Hexadecimal or Binary). ROM initialization is supported using memory-coefficient files. The data format is either binary or hexadecimal with each data entry on a new line in the memory-coefficient file (specified by parameter INIT_FILE). Supported file types are txt, mem, dat, and init (recommended).
6. Generate the ROM by clicking Generate, as described in *Specifying ROMs with SYNCore, on page 405* and add it to your design. All output files are in the directory you specified on page 1 of the wizard.

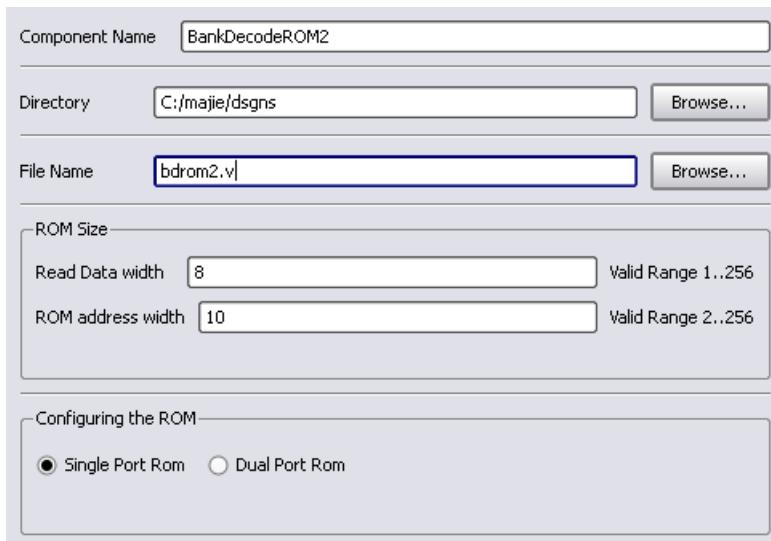
For read/write timing diagrams, see *Read/Write Timing Sequences, on page 383*.

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 411](#)
- [SYNCore ROM Parameters Pages 2 and 3, on page 412](#)
- [SYNCore ROM Parameters Page 4, on page 414](#)

SYNCORE ROM Parameters Page 1

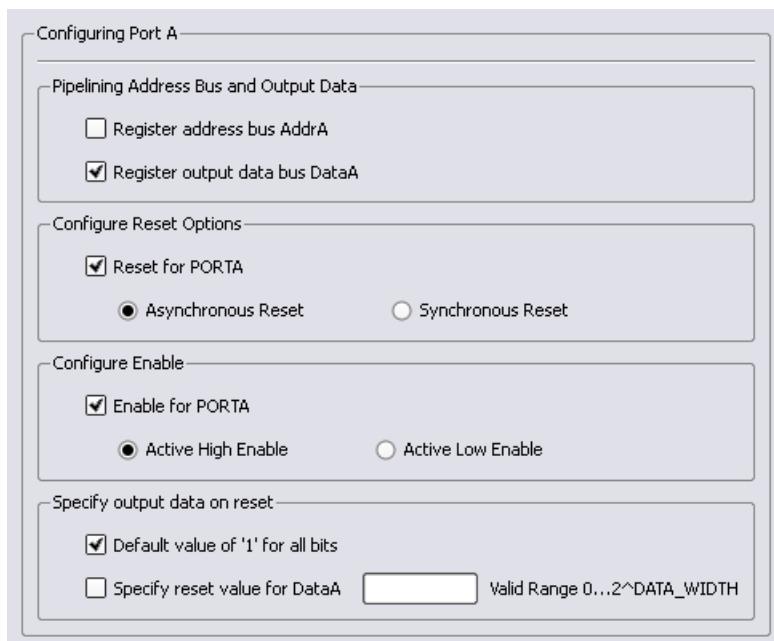


Component Name	Specifies the name of the component. This is the name that you instantiate in your design file to create an instance of the SYNCORE ROM in your design. Do not use spaces.
Directory	Specifies the directory where the generated files are stored. Do not use spaces. The following files are created: <code>filelist.txt</code> - lists files written out by SYNCORE <code>options.txt</code> - lists the options selected in SYNCORE <code>readme.txt</code> - contains a brief description and known issues <code>syncore_rom.v</code> - Verilog library file required to generate ROM model <code>testbench.v</code> - Verilog testbench file for testing the ROM model <code>instantiation_file.vin</code> - describes how to instantiate the wrapper file <code>component.v</code> - 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.

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".

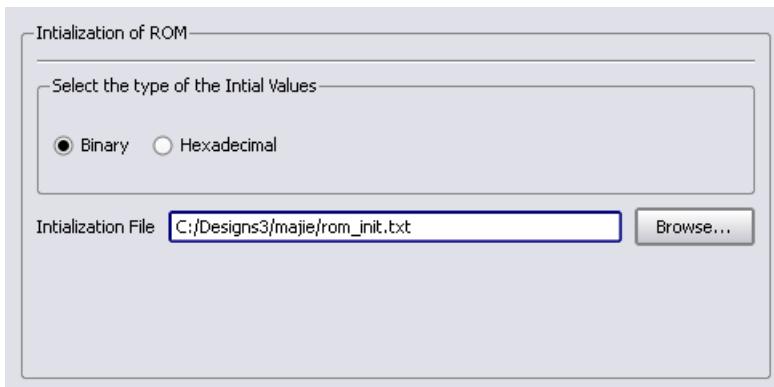
SYNCore ROM Parameters Pages 2 and 3

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.

SYNCore ROM Parameters Page 4

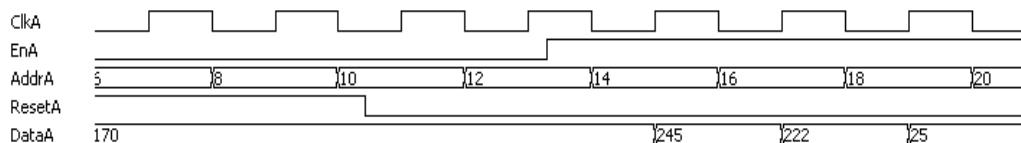


Binary	Specifies binary-formatted initialization file.
Hexadecimal	Specifies hexadecimal-formatted initial file.
Initialization File	Specifies path and filename of initialization file. The corresponding file parameter is INIT_FILE=" <i>filename</i> ".

Single-Port Read Operation

For single-port ROM, it is only necessary to configure Port A (see [Specifying ROMs with SYNCore, on page 405](#)). The following diagram shows the read timing for a single-port ROM.

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.

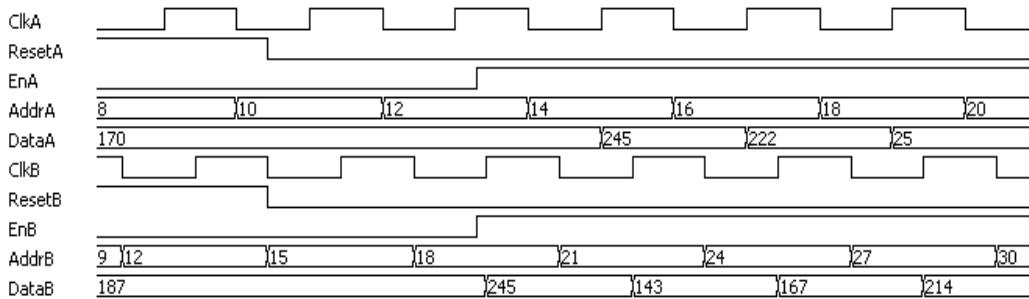


When reset is active, the output data holds the initialization value (i.e., 255). When reset goes inactive (and enable is active), data is read from the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset 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 417](#).

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 405](#)). 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 from 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 417](#).

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).
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 (Intel)	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 (Intel)	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

SYNCore Adder/Subtractor Compiler

The SYNCore adder/subtractor compiler generates Verilog code for a parameterizable, pipelined adder/subtractor. This section describes the functionality of this block in detail.

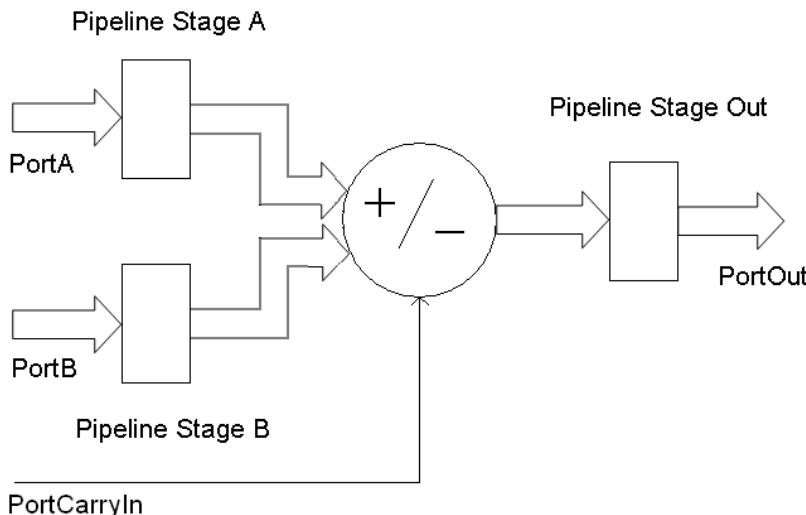
- [Functional Description](#), on page 418
- [Specifying Adder/Subtractors with SYNCore](#), on page 419
- [SYNCore Adder/Subtractor Wizard](#), on page 427
- [Adder](#), on page 430
- [Subtractor](#), on page 433
- [Dynamic Adder/Subtractor](#), on page 436

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

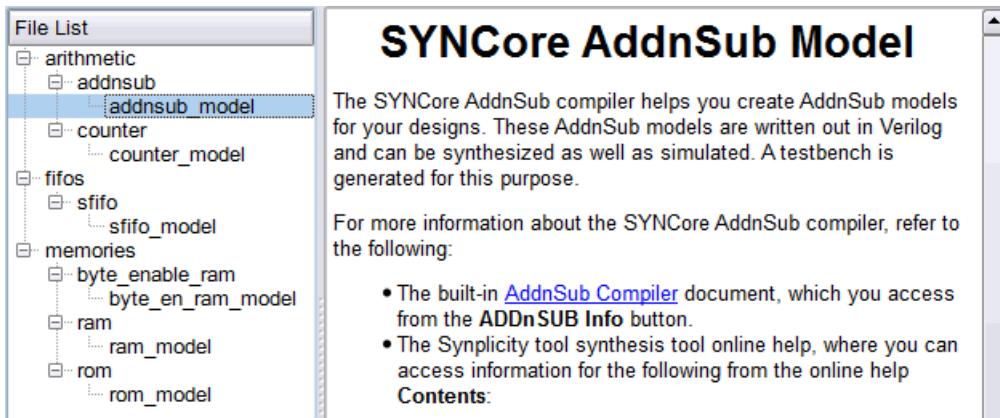
Specifying Adder/Subtractors with SYNCORE

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

Note: The SYNCORE adder/subtractor models use Verilog 2001. When adding an adder/subtractor model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a `set_option -vlog_std v2001` statement in your project file to prevent a syntax error.

1. Start the wizard.

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



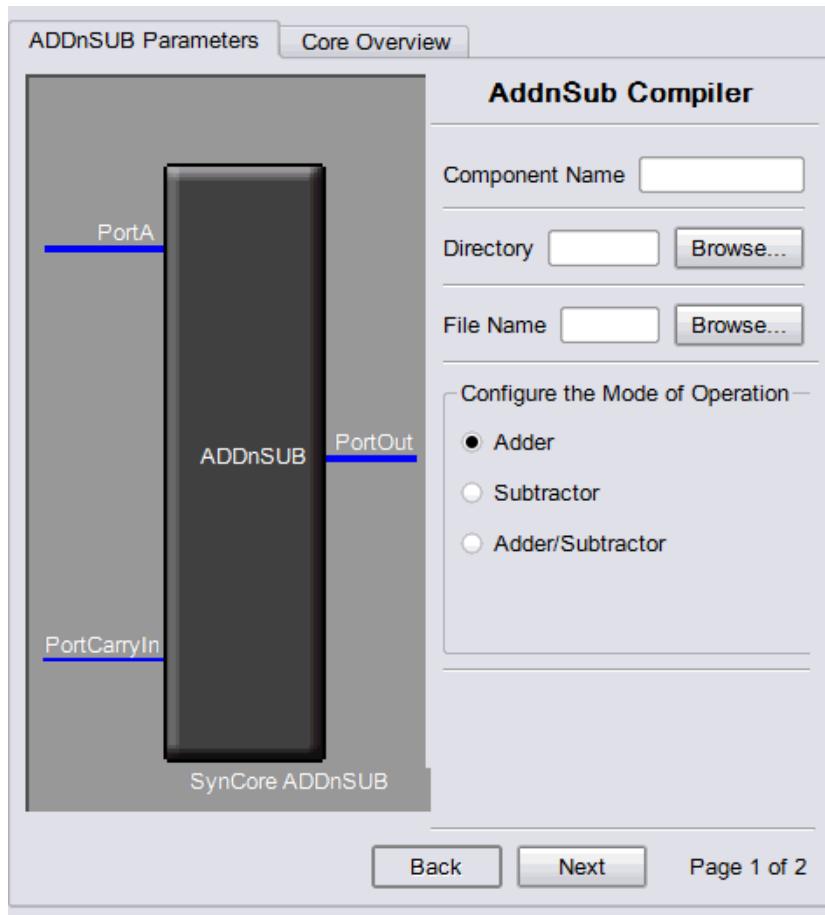
SYNCORE AddnSub Model

The SYNCORE AddnSub compiler helps you create AddnSub models for your designs. These AddnSub models are written out in Verilog and can be synthesized as well as simulated. A testbench is generated for this purpose.

For more information about the SYNCORE AddnSub compiler, refer to the following:

- The built-in [AddnSub Compiler](#) document, which you access from the ADDnSUB Info button.
- The Synplicity tool synthesis tool online help, where you can access information for the following from the online help [Contents](#):

- In the window that opens, select addnsb_model and click Ok to open page1 of the wizard.



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying Adder/Subtractor Parameters, on page 425](#). The ADDnSUB symbol on the left reflects any parameters you set.
3. After you have specified all the parameters you need, click the Generate button in the lower left corner.

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

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

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

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

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

endmodule
```

template

5. Edit the template port connections so that they agree with the port definitions in your top-level module as shown in the example below (the updated connection names are shown in red). You can also assign a unique name to each instantiation.

```
module top (
    output [15 : 0] Out,
    input Clk,
    input [15 : 0] A,
    input CEA,
```

```

    input RSTA,
    input [15 : 0] B,
    input CEB,
    input RSTB,
    input CEOut,
    input RSTOut,
    input ADDnSUB,
    input CarryIn);

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

```

Port List

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

Port Name	Description	Required/Optional
PortA	Data input for adder/subtractor Parameterized width and pipeline stages	Always present
PortB	Data input for adder/subtractor Parameterized width and pipeline stages	Not present if adder/subtractor is configured as a constant adder/subtractor
PortClk	Primary clock input; clocks all registers in the unit	Always present

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

Specifying Adder/Subtractor Parameters

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

- Adder
- Subtractor
- Dynamic Adder/Subtractor

If you are creating a constant input adder, subtractor, or a dynamic adder/subtractor with the SYNCore IP wizard, you must select Constant Value Input and specify a value for port B in the Constant Value/Port B Width field on page 2 of the parameters. The following procedure lists the parameters you need to define when generating an adder/subtractor. For descriptions of each parameter, see [SYNCore Adder/Subtractor Wizard, on page 427](#).

1. Start the SYNCORE adder/subtractor wizard as described in *Specifying Adder/Subtractors with SYNCORE*, on page 419.
2. Enter the following on page 1 of the wizard:
 - In the Component Name field, specify a name for your adder/subtractor. Do not use spaces.
 - In the Directory field, specify a directory where you want the output files to be written. Do not use spaces.
 - In the Filename field, specify a name for the Verilog file that will be generated with the adder/subtractor definitions. Do not use spaces.
 - Select the appropriate configuration in Configure the Mode of Operation.
3. Click Next. The wizard opens page 2 where you set parameters for port A and port B.
4. Configure Port A and B.
 - In the Configure Port A section, enter a value in the Port A Width field.
 - If you are defining a synchronous adder/subtractor, check Register Input A and then check Clock Enable for Register A and/or Reset for Register A.
 - To configure port B as a constant port, go to the Configure Port B section and check Constant Value Input. Enter the constant value in the Constant Value/Port B Width field.
 - To configure port B as a dynamic port, go to the Configure Port B section and check Enable Port B and enter the port width in the Constant Value/Port B Width field.
 - To define a synchronous adder/subtractor, check Register Input B and then check Clock Enable for Register B and/or Reset for Register B.
5. In the Configure Output Port section:
 - Enter a value in the Output port Width field.
 - If you are registering the output port, check Register output Port.
 - If you are defining a synchronous adder/subtractor check Clock Enable for Register PortOut and/or Reset for Register PortOut.
6. In the Configure Reset type for all Reset Signal section, click Synchronous Reset or Asynchronous Reset as appropriate.

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

7. Generate the adder/subtractor by clicking the Generate button as described in [Specifying Adder/Subtractors with SYNCore, on page 419](#) and add it to your design. All output files are in the directory you specified on page 1 of the wizard.

SYNCore Adder/Subtractor Wizard

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

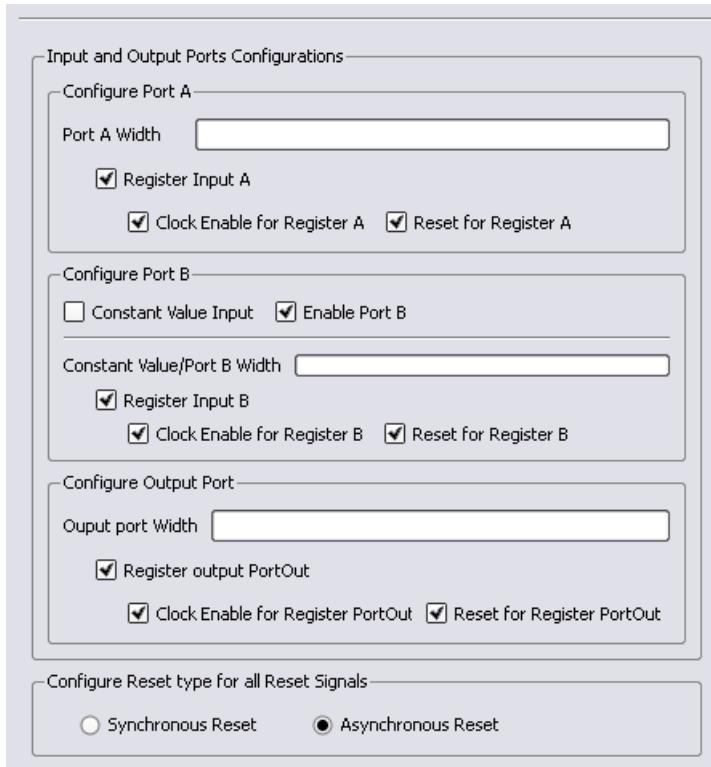
- [SYNCore Adder/Subtractor Parameters Page 1, on page 427](#)
- [SYNCore Adder/Subtractor Parameters Page 2, on page 429](#)

SYNCore Adder/Subtractor Parameters Page 1

The screenshot shows a software interface for configuring a SYNCore Adder/Subtractor. It includes fields for Component Name (set to "adder_subtractor3"), Directory (set to "C:/designs/majie"), and File Name (set to "add_sub4"). Below these, there is a section titled "Configure the Mode of Operation" containing three radio buttons: "Adder", "Subtractor", and "Adder/Subtractor". The "Adder/Subtractor" option is selected.

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	<p>Indicates the directory where the generated files will be stored. Do not use spaces. The following files are created:</p> <ul style="list-style-type: none"> • <code>filelist.txt</code> - lists files written out by SYNCORE • <code>options.txt</code> - lists the options selected in SYNCORE • <code>readme.txt</code> - contains a brief description and known issues • <code>syncore_ADDnSUB.v</code> - Verilog library file required to generate adder/subtractor model • <code>testbench.v</code> - Verilog testbench file for testing the adder/subtractor model • <code>instantiation_file.vin</code> - describes how to instantiate the wrapper file • <code>component.v</code> - adder/subtractor model wrapper file generated by SYNCORE <p>Note that running the wizard in the same directory overwrites any existing files.</p>
Filename	Specifies the name of the generated file containing the HDL description of the generated adder/subtractor. Do not use spaces.
Adder	When enabled, generates an adder (the corresponding file parameter is <code>ADD_N_SUB = "ADD"</code>).
Subtractor	When enabled, generates a subtractor (the corresponding file parameter is <code>ADD_N_SUB = "SUB"</code>).
Adder/Subtractor	When enabled, generates a dynamic adder/subtractor (the corresponding file parameter is <code>ADD_N_SUB = "DYNAMIC"</code>).

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.
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').

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.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTB_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '0'

PortClk						
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 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'
```

PortClk	
PortA	0 4 9 13 5 1
PortB	0 1 3 5 2 5
PortOut	0 5 12 18 7

Adder with a Port Constant

In this mode, port A is added with a constant value (the constant value can be passed through 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.

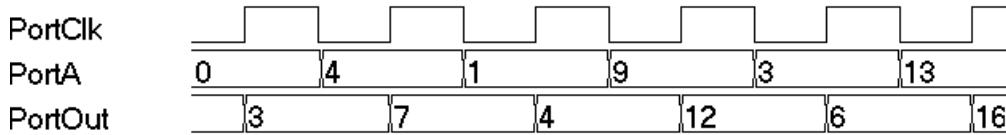
Parameters:

```
PORTA_PIPELINE_STAGE='0'
PORTOUT_PIPELINE_STAGE='0'
```

PortA	0 4 1 9 3 13
PortOut	3 7 4 12 6 16

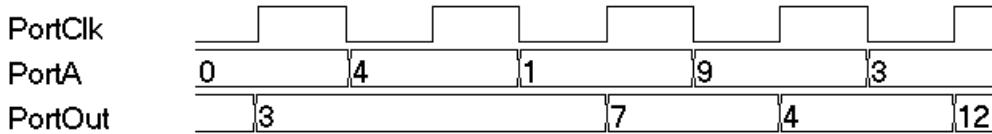
Adder with Pipeline Stage at Input Only - In this mode, input port A is pipelined and added with a constant value. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

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'



Subtractor

Based on the parameter CONSTANT_PORT, the subtractor can be configured 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:

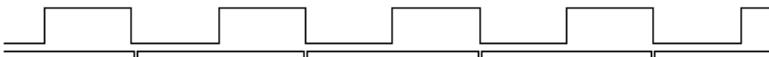
- PORTA_PIPELINE_STAGE= '0'
- PORTB_PIPELINE_STAGE= '0'
- PORTOUT_PIPELINE_STAGE= '0'

PortA	0	4	9	13	5
PortB	0	1	3	5	2
PortOut	0	3	6	8	3

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:

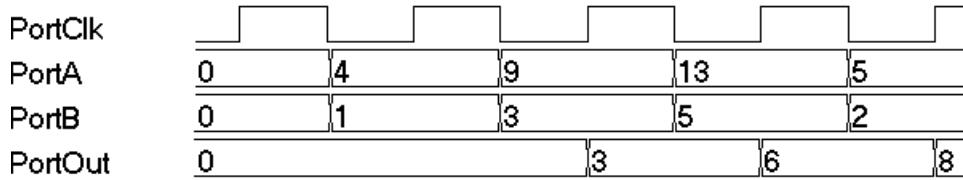
- PORTA_PIPELINE_STAGE= '1'
- PORTB_PIPELINE_STAGE= '1'
- PORTOUT_PIPELINE_STAGE= '0'

PortClk					
PortA	0	4	9	13	5
PortB	0	1	3	5	2
PortOut	0	3	6	8	3

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:

- PORATA_PIPELINE_STAGE= '1'
- PORTB_PIPELINE_STAGE= '1'
- PORTOUT_PIPELINE_STAGE= '1'



Subtractor with a Port Constant

In this mode, a constant value is subtracted from port A (the constant value can be passed through 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.

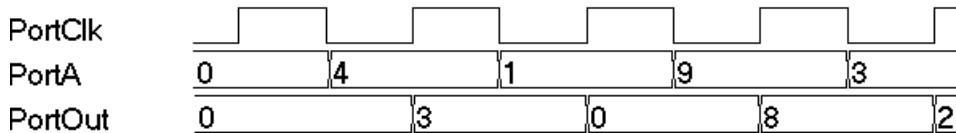
Parameters:

- PORATA_PIPELINE_STAGE= '0'
- PORTOUT_PIPELINE_STAGE= '0'

PortA	0	4	1	9	3
PortOut	0	3	0	8	2

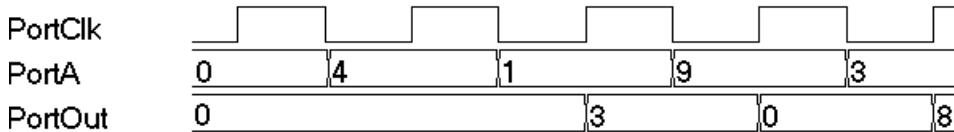
Subtractor with Pipeline Stages at Input Only - In this mode, a constant value is subtracted from pipelined input port A. Because there is no pipeline stage at the output, the output is valid at each rising edge of the clock.

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.

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.

Parameters:

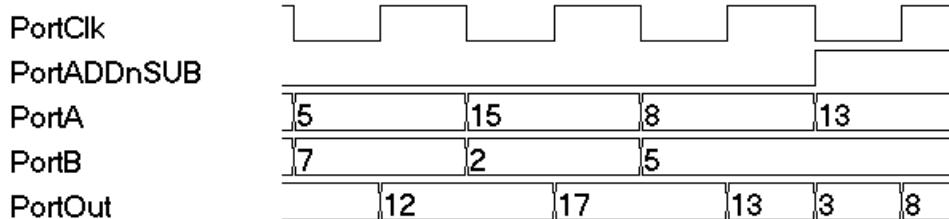
PORTA_PIPELINE_STAGE= '0'
PORTB_PIPELINE_STAGE= '0'
PORTOUT_PIPELINE_STAGE= '0'

PortADDnSUB					
PortA	5	15	8	13	
PortB	7	2	5		
PortOut	12	17	13	8	

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:

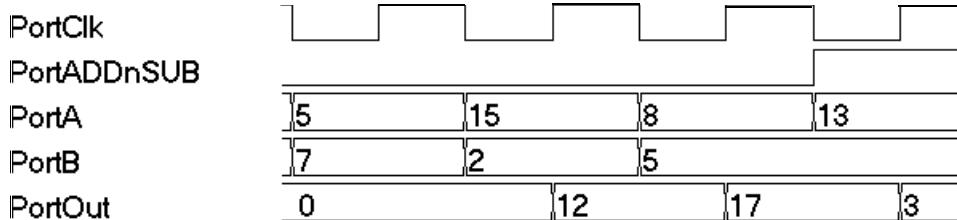
PORTA_PIPELINE_STAGE= '1'
PORTB_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '0'



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.

Parameters:

```
PORATA_PIPELINE_STAGE='1'  
POROTB_PIPELINE_STAGE='1'  
POROUT_PIPELINE_STAGE='1'
```



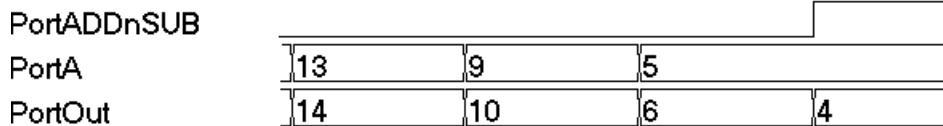
Dynamic Adder/Subtractor with a Port Constant

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 through 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:

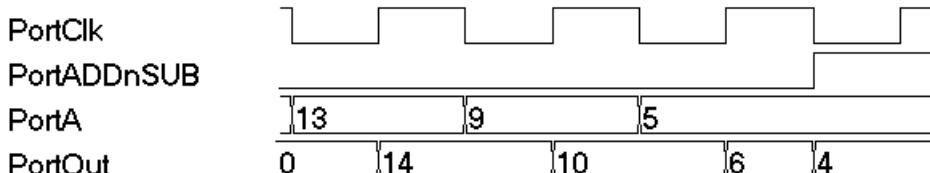
- PORTA_PIPELINE_STAGE= '0'
- PORTOUT_PIPELINE_STAGE= '0'



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.

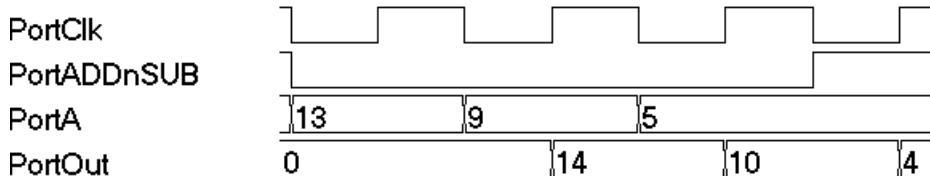
Parameters:

- PORTA_PIPELINE_STAGE= '1'
- PORTOUT_PIPELINE_STAGE= '0'



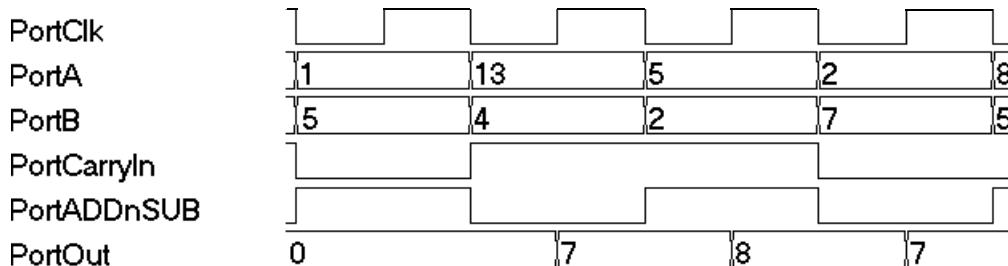
Dynamic Adder/Subtractor with Pipeline Stages at Input and Output - In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because the output port is pipelined, the result is valid only on the second rising edge of the clock.

Parameters:
 PORTA_PIPELINE_STAGE= '1'
 PORTOUT_PIPELINE_STAGE= '1'



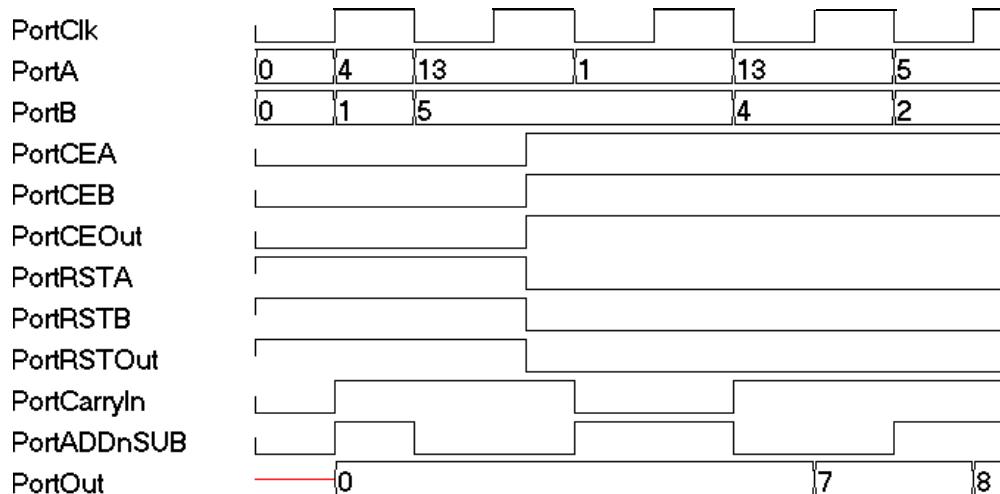
Dynamic Adder/Subtractor with Carry Input

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.



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 442
- [Specifying Counters with SYNCore](#), on page 443
- [SYNCore Counter Wizard](#), on page 449
- [UP Counter Operation](#), on page 452
- [Down Counter Operation](#), on page 453
- [Dynamic Counter Operation](#), on page 453

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 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.

Note: Intel 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.

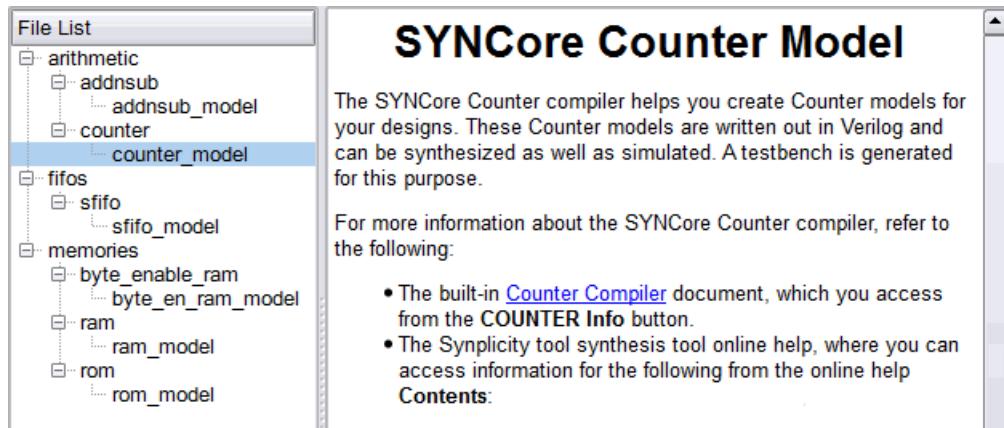
Specifying Counters with SYNCore

The SYNCore IP wizard helps you generate Verilog code for your counter implementation requirements. The following procedure shows you how to generate Verilog code for a counter using the SYNCore IP wizard.

Note: The SYNCore counter model uses Verilog 2001. When adding a counter model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a `set_option -vlog_std v2001` statement in your project file to prevent a syntax error.

1. Start the wizard.

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



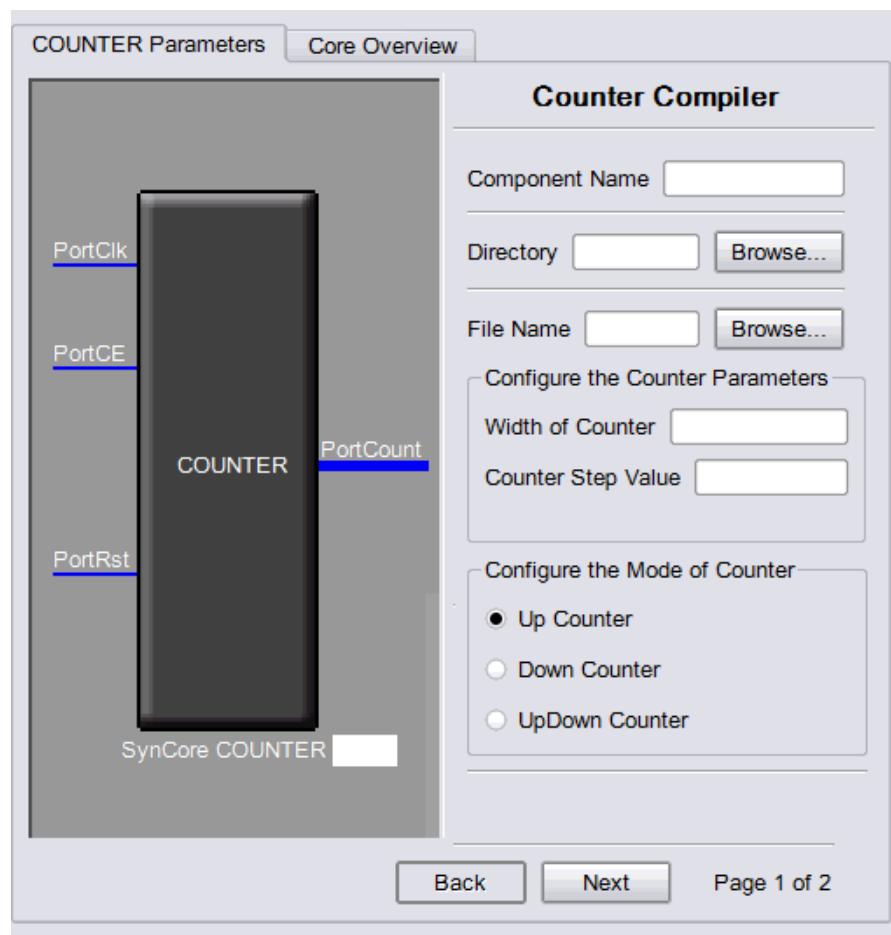
SYNCore Counter Model

The SYNCore Counter compiler helps you create Counter models for your designs. These Counter models are written out in Verilog and can be synthesized as well as simulated. A testbench is generated for this purpose.

For more information about the SYNCore Counter compiler, refer to the following:

- The built-in [Counter Compiler](#) document, which you access from the COUNTER Info button.
- The Synplicity tool synthesis tool online help, where you can access information for the following from the online help Contents:

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



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying Counter Parameters, on page 448](#). The COUNTER symbol on the left reflects any parameters you set.
3. After you have specified all the parameters you need, click the Generate button in the lower left corner.

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

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

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

```
module counter #(  
    parameter COUNT_WIDTH = 5,  
    parameter STEP = 2,  
    parameter RESET_TYPE = 0,  
    parameter LOAD = 2,  
    parameter MODE = "Dynamic")  
{  
    // Output Ports  
    output wire [WIDTH-1:0] Count,  
    // Input Ports  
    input wire Clock,  
    input wire Reset,  
    input wire Up_Down,  
    input wire Load,  
    input wire [WIDTH-1:0] LoadValue,  
    input wire Enable );  
  
SynCoreCounter #(  
    .COUNT_WIDTH(COUNT_WIDTH),  
    .STEP(STEP),  
    .RESET_TYPE(RESET_TYPE),  
    .LOAD(LOAD),  
    .MODE(MODE))  
SynCoreCounter_ins1 (  
    .PortCount(PortCount),  
    .PortClk(PortClock),  
    .PortRST(PortRST),  
    .PortUp_nDown(PortUp_nDown),  
    .PortLoad(PortLoad),  
    .PortLoadValue(PortLoadLoadValue)  
    .PortCE(PortCE) );  
template  
endmodule
```

5. Edit the template port connections so that they agree with the port definitions in your top-level module as shown in the example below (the updated connection names are shown in red). You can also assign a unique name to each instantiation. You can also assign a unique name to each instantiation.

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

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

SynCoreCounter_ins1 (
    .PortCount(Count),
    .PortClk(Clock),
    .PortRST(Reset),
    .PortUp_nDown(Up_Down),
    .PortLoad(Load),
    .PortLoadValue(LoadValue),
    .PortCE(Enable));

endmodule
```

Port List

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

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

Specifying Counter Parameters

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

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

The counter core can have a constant or variable input load or no load value. If you are creating a constant-load counter, you will need to select **Enable Load** and **Load Constant Value** on page 2 of the wizard. If you are creating a variable-load counter, you will need to select **Enable Load** and **Use Variable Port Load** on page 2. The following procedure lists the parameters you need to define when generating a counter. For descriptions of each parameter, see [SYNCORE Counter Wizard, on page 449](#).

1. Start the SYNCORE counter wizard, as described in [Specifying Counters with SYNCORE, on page 443](#).

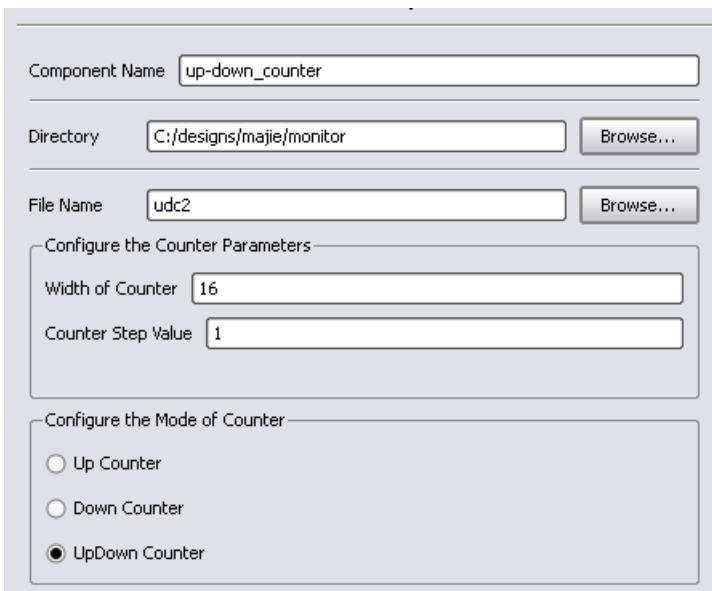
2. Enter the following on page 1 of the wizard:
 - In the Component Name field, specify a name for your counter. Do not use spaces.
 - In the Directory field, specify a directory where you want the output files to be written. Do not use spaces.
 - In the Filename field, specify a name for the Verilog file that will be generated with the counter definitions. Do not use spaces.
 - Enter the width and depth of the counter in the Configure the Counter Parameters section.
 - Select the appropriate configuration in the Configure the Mode of Counter section.
3. Click Next. The wizard opens page 2 where you set parameters for PortLoad and PortLoadValue.
 - Select Enable Load option and the required load option in Configure Load Value section.
 - Select the required reset type in the Configure Reset type section.The COUNTER symbol dynamically updates to reflect the parameters you set.
4. Generate the counter core by clicking Generate button. All output files are written to the directory you specified on page1 of the wizard.

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 450](#)
- [SYNCORE Counter Parameters Page 2, on page 451](#)

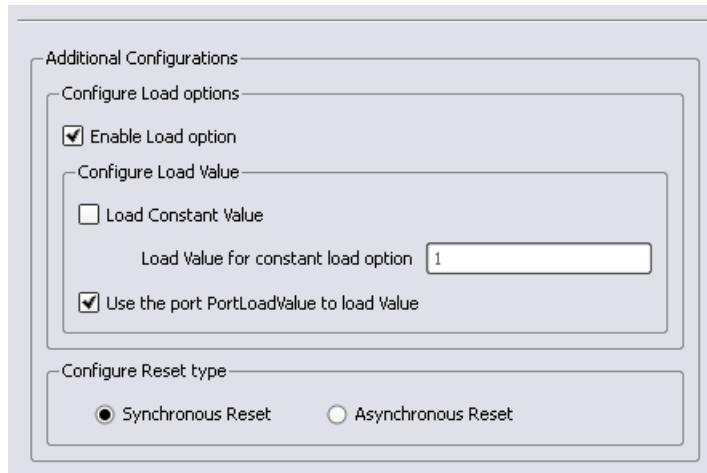
SYNCore Counter Parameters Page 1



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	<p>Indicates the directory where the generated files will be stored. Do not use spaces. The following files are created:</p> <ul style="list-style-type: none"> • 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 • <i>component.v</i> - counter model wrapper file generated by SYNCore <p>Note that running the wizard in the same directory overwrites any existing files.</p>
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).

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.

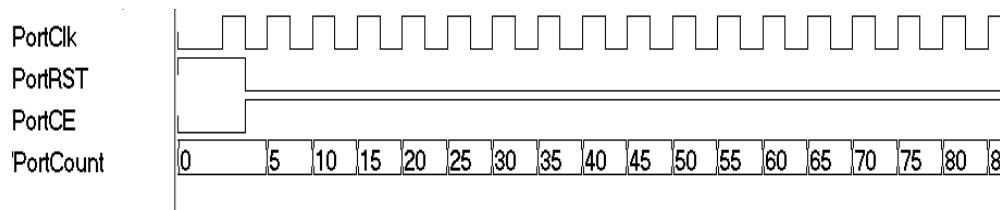
Use the port PortLoadValue to load Value	Loads variable value from PortLoadValue (the corresponding file parameter is LOAD=2).
Synchronous Reset	Specifies a synchronous (the default) reset input (the corresponding file parameter is MODE=0).
Asynchronous Reset	Specifies an asynchronous reset input (the corresponding file parameter is MODE=1).

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'

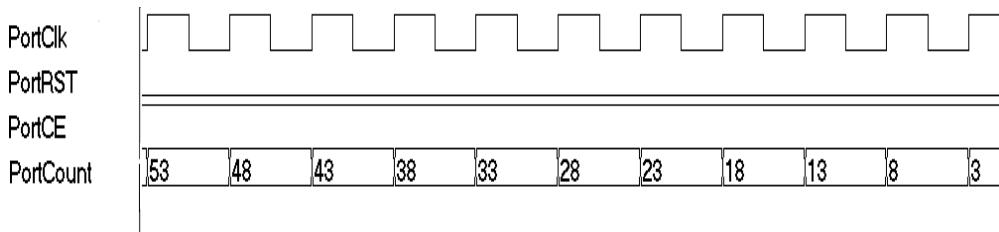


Note: Counter core can be configured to use a constant or dynamic load value in Up Counter mode (for the counter to load the PortLoadValue, PortCE must be active). This functionality is explained in [Dynamic Counter Operation, on page 453](#).

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 453](#).

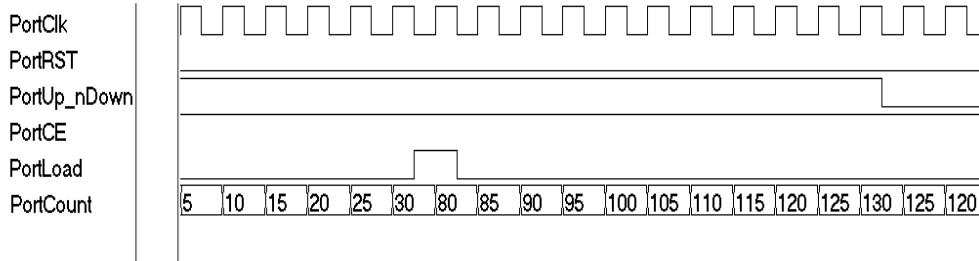
Dynamic Counter Operation

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.

Parameters: MODE= 'Dynamic'
 LOAD= '1'



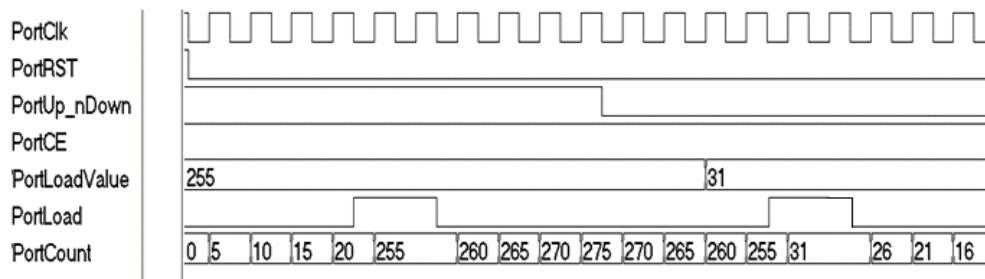
Note: *For counter to load the PortLoadValue, PortCE must be active.

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'



Note: * For counter to load the PortLoadValue, PortCE should be active.

CHAPTER 8

Physical Optimization Tools

The Synplify Premier physical optimization 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 Design Planner and Physical Analyst views are presented here:

- [Design Planner User Interface](#), on page 458
- [Physical Analyst User Interface](#), on page 464

For details about the standard windows, views, and menu commands for all the Synopsys FPGA products, see also:

- [Chapter 2, User Interface Overview](#)
- [Chapter 5, User Interface Commands](#)

Design Planner User Interface

This section describes the graphical user interface (GUI) for the Synplify Premier Design Planner tool. See [Design Planner View, on page 458](#).

For the Design Planner commands and menus, see the following topics:

- [Design Planner UI Commands](#), on page 658
- [Design Planner Tools Menu](#), on page 667
- [Design Planner Popup Menus](#), on page 673

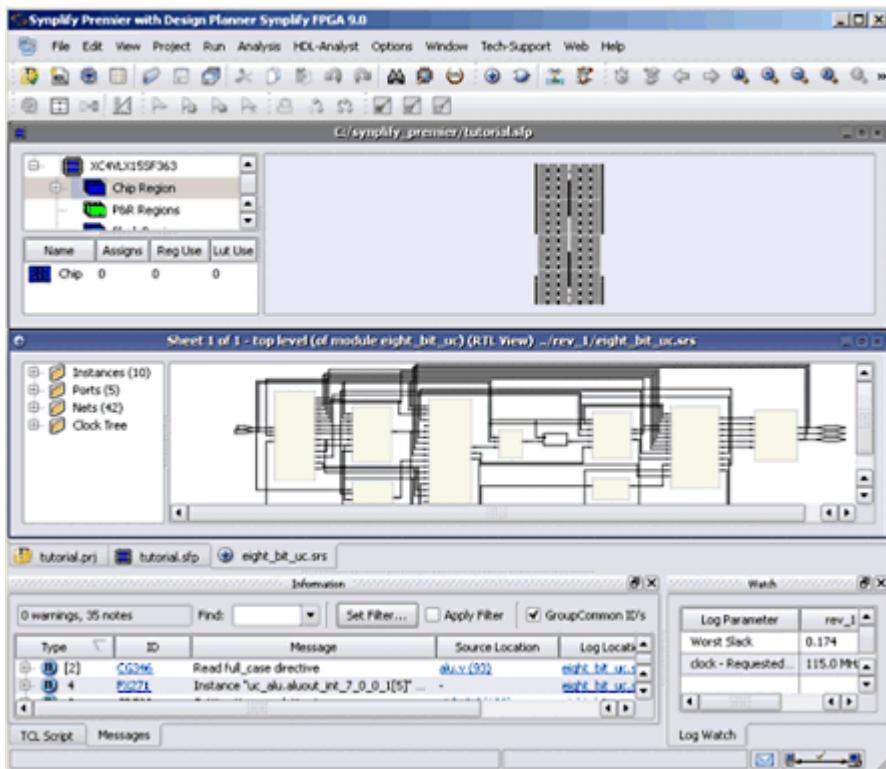
Design Planner View

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 the [RTL View, on page 98](#). The other Design Planner components are described here:

- [Design Plan Hierarchy View](#), on page 459
- [Design Plan View](#), on page 461
- [Design Plan Editor View](#), on page 461

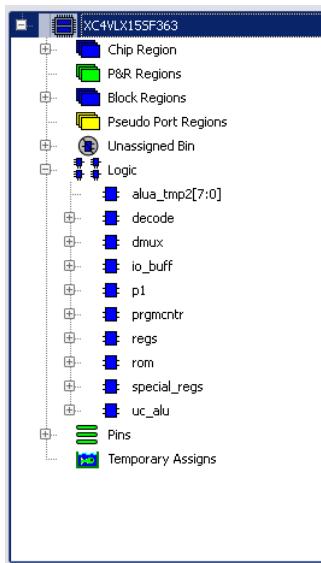
For more information about using this tool, refer to [Using Design Planner, on page 998](#) and for a description of how to create regions, see [Design Planner - Working with Regions, on page 1014](#) in the *User Guide*.

Design Plan View



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 the “+” and “-” signs to expand and contract the hierarchy.



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 459](#).

Name	Assigns
rgn1	82
rgn2	1

Default View

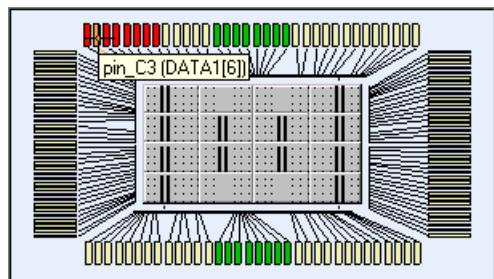
Name	S.	View	As...	Co...
DECODE_AL...	0	pmux	rgn1	14
DECODE.pr...	24	INS_Decod...	rgn1	50
DECODE.un...	0	or	rgn1	4
DECODE.un...	0	or	rgn1	4
DECODE.un...	0	or	rgn1	4
DECODE.un...	0	or	rgn1	15
DECODE.un...	0	or	rgn1	4
DECODE.un...	0	and	rgn1	3
DECODE.un...	0	and	rgn1	3
DECODE.un...	0	and	rgn1	13
DECODE.un...	0	or	rgn1	8
DECODE.un...	0	inv	rgn1	2
DECODE.un...	0	and	rgn1	3
DECODE.un...	0	and	rgn1	3
DECODE.un...	0	and	rgn1	3
DECODE.un...	0	and	rgn1	3
DECODE.un...	0	or	rgn1	3

Context Sensitive View

Design Plan Editor View

The Design Plan Editor view shows the physical layout of the specified device and the placement of constraints.

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 1001](#) 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:

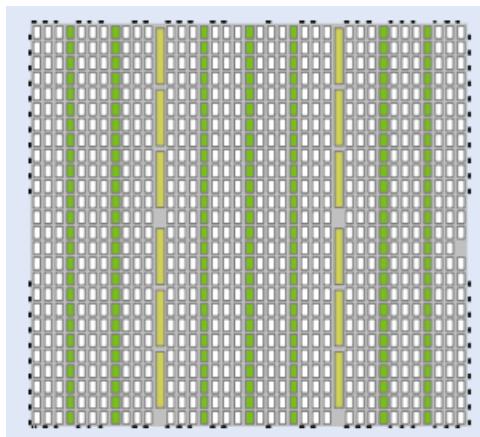
Orange	Selected assigned pins
Red	Deselected assigned pins
Blue	Selected unassigned pins

Device Display

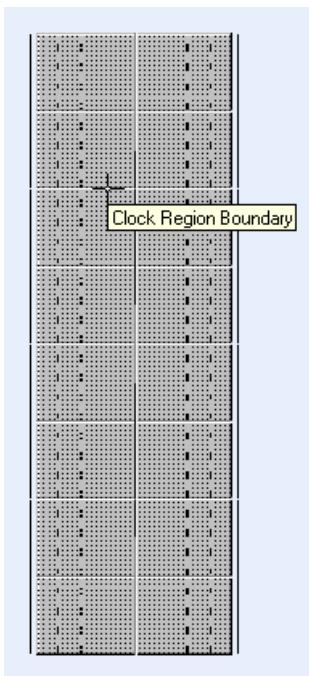
Depending on the technology specified, the image displays LAB (Intel technology) or CLB (Xilinx technology) structures and device-sensitive resources (for example, RAMs or DSPs). The view is context sensitive, so the information displayed in the pane depends on your settings and the objects selected 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 Intel device.



The following figure shows the Design Plan Editor view for a Xilinx device. Note the clock region boundary.



Physical Analyst User Interface

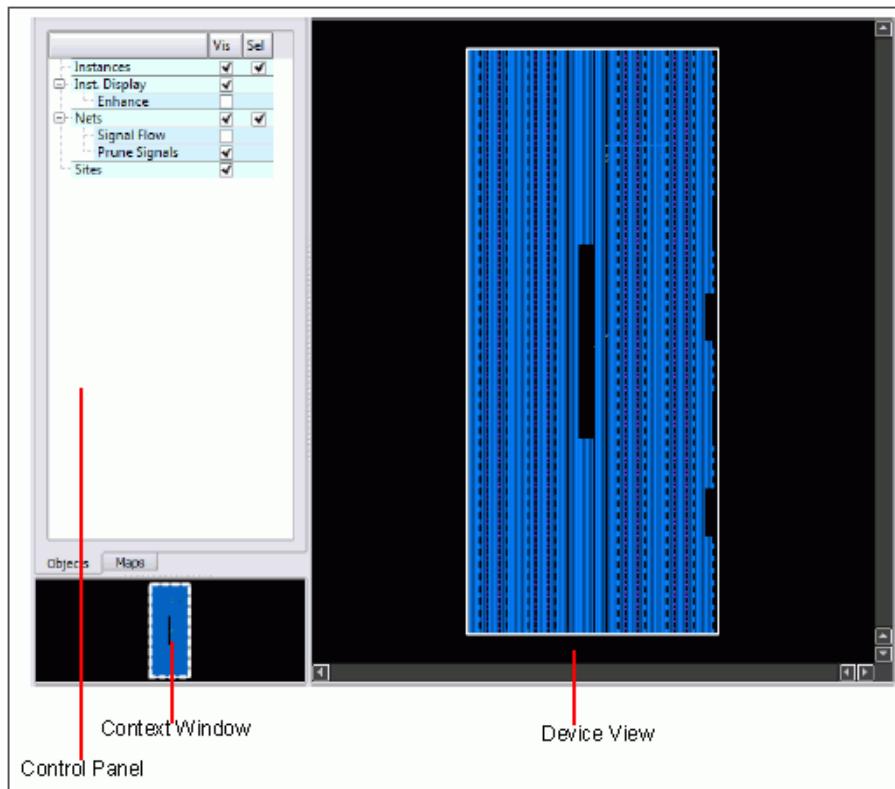
This section describes the graphical user interface (GUI) for the Synplify Premier Physical Analyst tool. See [Physical Analyst User Interface, on page 464](#).

For the Physical Analyst commands and menus, see the following topics:

- [Physical Analyst UI Commands](#), on page 686
- [Physical Analyst View Popup Menus](#), on page 692

Physical Analyst View

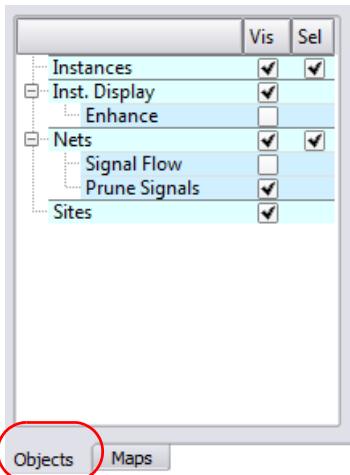
The Physical Analyst tool provides a visual display of the placement and global routing of the design after running synthesis and place and route. It consists of an optional control panel and a window that displays a device.



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 (Ξ) 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 469](#).



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 1051 .
Instance Display	You can: <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> 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).

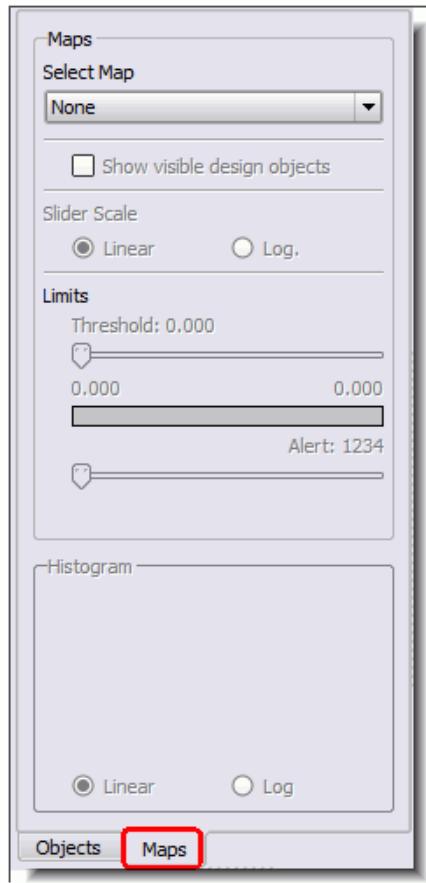
The device display in the Physical Analyst view is based on the following input files:

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 1088](#).



Command	Description
Select Maps	You can choose one of the following Implementation Maps: <ul style="list-style-type: none">Route CongestionBlock Component UtilizationBlock Input UtilizationSlack DistributionNone—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

APPENDIX A

Designing with Achronix

This section contains guidelines for synthesizing Achronix designs. It discusses the following topics:

- [Basic Support for Achronix Designs](#), on page 472
- [Achronix Components](#), on page 475
- [Achronix Device Mapping Options](#), on page 527
- [Achronix Output Files and Forward Annotation](#), on page 534
- [Integration with Achronix Tools and Flows](#), on page 540
- [Achronix Attribute and Directive Summary](#), on page 542

Basic Support for Achronix Designs

This section describes general guidelines for using the synthesis tool and Achronix place-and-route (P&R) tool with Achronix devices. Refer to:

- [Achronix Speedster7t Device-specific Support](#), on page 472
- [Supported Device Families](#), on page 473
- [Netlist Format](#), on page 473

Achronix Speedster7t Device-specific Support

This section describes the Achronix Speedster7t device features:

- Support for asymmetric RAM inference using BRAM72K_SD_P primitive with the Speedster7t technology.
- Support to map symmetric RAM, Read only memories using new BRAM72K_SD_P and LRAM2K_SD_P for Speedster7t technology.
- RAM resource management using BRAM72K_SD_P and LRAM2K_SD_P for Speedster7t technology.
- Wide modes (X128, X144) for BRAM72K_SD_P primitive is supported with syn_ramstyle = large_ram and syn_romstyle= large_rom attributes.
- MLP inference enhancements for Speedster7t technology:
 - Direct input Multiplier/Subtractor inference using MLP72_INT is supported.
 - $y = a*b + c*d + e*f + g*h$ function inference is supported with all inputs up to 8 bits.
 - Map multiplier functions using MLP primitives.
 - Map multiplier accumulator function ($y = a*b + y$) up to 16*16 bits using MLP72_INT.
 - Map multiplier adder ($y = a*b + c*d$) function up to 16*16 bits using MLP72_INT.
- RAM inference report:
 - Synplify tool generates a RAM inference report (design_name_ram_rpt.txt) for Speedster7t technology. The RAM inference report includes the RAM type, attribute usage, width and depth configuration, and the

reasons for mapping RTL RAMs to Block RAM, LRAM, and registers. The report also includes the scenarios for instances when the Synplify tool reports any errors during the RAM mapping.

Supported Device Families

The synthesis tool creates technology-specific netlists for the following Achronix technologies:

- Achronix Speedster7t

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.

Netlist Format

The synthesis tool outputs VM netlist files for use with the Achronix place-and-route application. These files have a `.vm` extension.

You can also use the project Tcl command to specify the result file format.

```
project -result_format vm
```

To generate Achronix output, see [Targeting Output for Achronix, on page 473](#).

Targeting Output for Achronix

You can generate output targeted for Achronix.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

The following table summarizes the output to set for the different devices, and shows the P&R tool for which the output is intended.

Vendor Support	Output Netlist	P&R Tool
Achronix	VM (.vm)	ACE

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

Achronix Forward Annotation

The synthesis tool generates Achronix-compliant constraint files from selected constraints that are forward annotated (read in and then used) by the Achronix ACE place-and-route software. The ACE constraint file uses the `.scf` extension. This constraint file must be imported into the Achronix flow.

By default, Achronix 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).

Achronix Components

The following topics describe how the synthesis tool handles various Achronix components, and shows you how to work with or manipulate them during synthesis to get the results you need:

- [Achronix Asymmetric RAM](#), on page 475
- [Achronix LRAM2K_SDPI Inference](#), on page 485
- [Packing Output Registers in the LRAM2K_SDPI](#), on page 489
- [Achronix BRAM72K_SDPI Inference](#), on page 490
- [Achronix ROMs](#), on page 491
- [RAM and ROM Initialization for Achronix](#), on page 495
- [Achronix MLP primitives Inference](#), on page 495
- [Achronix Latch Inference](#), on page 497
- [Achronix Register Inference](#), on page 511
- [Register Initialization for Achronix](#), on page 511
- [Achronix Shift Register Inference](#), on page 523

Achronix Asymmetric RAM

The synthesis software supports asymmetric simple dual-port RAMs. The following asymmetric RAM configurations are supported:

- [Asymmetric RAM with Read Width > Write Width](#), on page 475
- [Asymmetric RAM with Write Width > Read Width](#), on page 479

For limitations, see [Asymmetric RAM Limitations](#), on page 482.

Asymmetric RAM with Read Width > Write Width

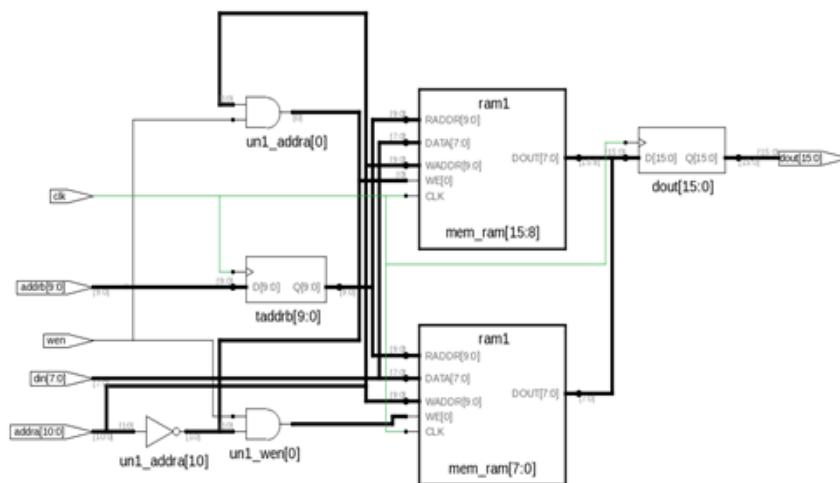
In this mode, the RAM read width is an integer multiple of the write width. The compiler creates K sub-instances for the RAM with `min_depth` and `min_width`, which can be combined and mapped into BRAM.

Example 1 (Read Width > Write Width): Coding Style 1

The following example displays the coding style for a MSB ordered asymmetric RAM (K=2) with write width < read width. In the RTL below, the write access configuration is 2Kx8 and the read access configuration is 1Kx16.

```
module asym_ram (din, dout, addra, addrb, clk, wen);
    input [7:0] din;
    input wen;
    input [10:0] addra;
    output reg [15:0] dout;
    input [9:0] addrb;
    input clk;
    localparam ratio= 2;
    localparam max_depth=2048;
    localparam min_width=8;
    reg [10:0] taddra;
    reg [min_width-1:0] mem_ram[max_depth-1:0];
    always @ (posedge clk)
    begin
        if (wen)
            mem_ram[taddra]<=din;
            taddra<=addra;
    end

    always @ (posedge clk)
    begin // manual concatenation
        dout[min_width*0+min_width]<=mem_ram[{0,addrb}];
        // it can be written inside generate-loop
        dout[min_width*1+min_width]<=mem_ram[{1,addrb}];
    end
endmodule
```



2Kx8 Write and 1Kx16 Read Asymmetric RAM

Resource utilization for 2Kx8 write and 1Kx16 read asymmetric RAM contains:

- BRAM 1 (read width=16, write width=8)
- DFF 10 (read address register)

Example 2 (Read Width > Write Width): Coding Style 2

The example below displays the coding style for LSB ordered asymmetric RAM ($K=2$) with write width < read width. In the RTL below, the write access configuration is 2Kx8 and the read access configuration is 1Kx16.

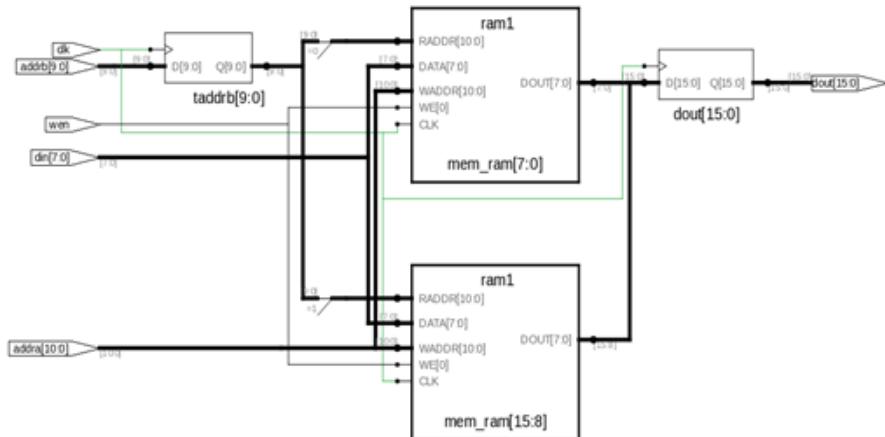
```
module asym_ram (din,dout, addra, addrb, clk, wen);
  input [7:0] din;
  input wen;
  input [10:0] addra;
  output reg [15:0] dout;
  input [9:0] addrb;
  input clk;
  localparam ratio= 2;
  localparam max_depth=2048;
  localparam min_width=8;
  reg [10:0] taddr;
  reg [min_width-1:0] mem_ram[max_depth-1:0];
```

```

always @ (posedge clk)
begin
    if(wen)
        mem_ram[taddrb]<=din;
        taddrb<=addrb;
end

always @ (posedge clk)
begin // manual concatenation
    dout[min_width*0+:min_width]<=mem_ram[{addrb,1'b0}];
    // it can be written inside generate-loop
    dout[min_width*1+:min_width]<=mem_ram[{addrb,1'b1}];
end
endmodule

```



2Kx8 write-1Kx16 read asymmetric RAM which is mapped to 1 BRAM

Resource utilization for 2Kx8 write and 1Kx16 read asymmetric RAM contains:

- BRAM 1 (read width=16, write width=8)
- DFF 10 (read address register)

Asymmetric RAM with Write Width > Read Width

The software provides the following methods for defining an asymmetric RAM for Achronix devices:

- [Asymmetric RAM Using NRAM](#), on page 479
- [Asymmetric RAM Using Symmetric RAM](#), on page 480

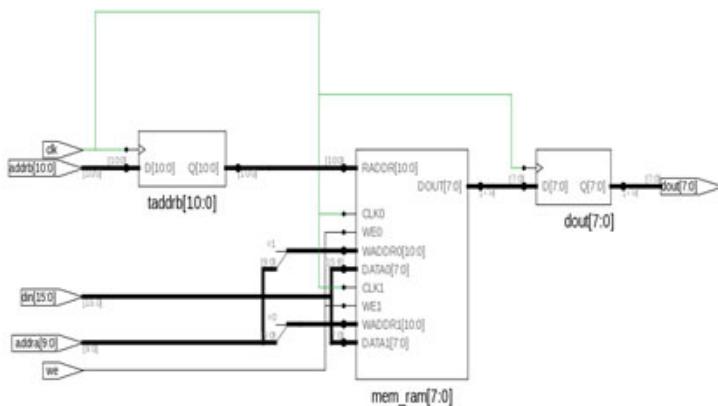
Asymmetric RAM Using NRAM

With this method, the compiler generates an NRAM containing multiple write processes for each of the K subsections of the RAM. The NRAM is mapped to BRAM.

Example 3 (Write Width > Read Width): Coding Style 1

The example below displays the coding style for an asymmetric RAM with K=2 when write width > read width. The code below implements asymmetric RAM with 1Kx16 write access and 2Kx8 read access configuration.

```
module asym_ram(din, dout, addra, addrb, clk, we);
    input [15:0] din;
    input [9:0] addra;
    output reg [7:0] dout;
    input [10:0] addrb;
    input clk;
    input we;
    localparam max_depth=2048;
    localparam min_width=8;
    reg [min_width-1:0] mem_ram[max_depth-1:0];
    reg [9:0] taddra;
    reg [10:0] taddrb;
    always @(posedge clk)
    begin
        dout<=mem_ram[taddrb];
        taddrb<=addrb;
    end
    always @(posedge clk)
    if (we)
    begin
        mem_ram[{0,addra}]<=din[min_width*0+:min_width];
        mem_ram[{1,addra}]<=din[min_width*1+:min_width];
    end
endmodule
```



2Kx8 read-1Kx16 write asymmetric RAM which is mapped to 1 BRAM

Resource utilization for 2Kx8 read and 1Kx16 write asymmetric RAM contains:

- BRAM 1 (read width=8, write width=16)
- DFF 11 (read address register)

Asymmetric RAM Using Symmetric RAM

With this method, an asymmetric RAM with write width > read width is implemented using a symmetric RAM and MUX at the output to create the asymmetric RAM behavior.

Example 4 (Write Width > Read Width): Coding Style 2

In the example below, the asymmetric RAM with 1Kx32 write mode and 4Kx8 read mode was created using a symmetric RAM of size 1Kx32 and output MUX4:1.

```
module aram (rst,din,dout,addra,addrb,clkr,clkw,wen,ren);
parameter rwidth=8;
parameter wwidth=32;
parameter wdepth=1024;
parameter rdepth=4*1024;
```

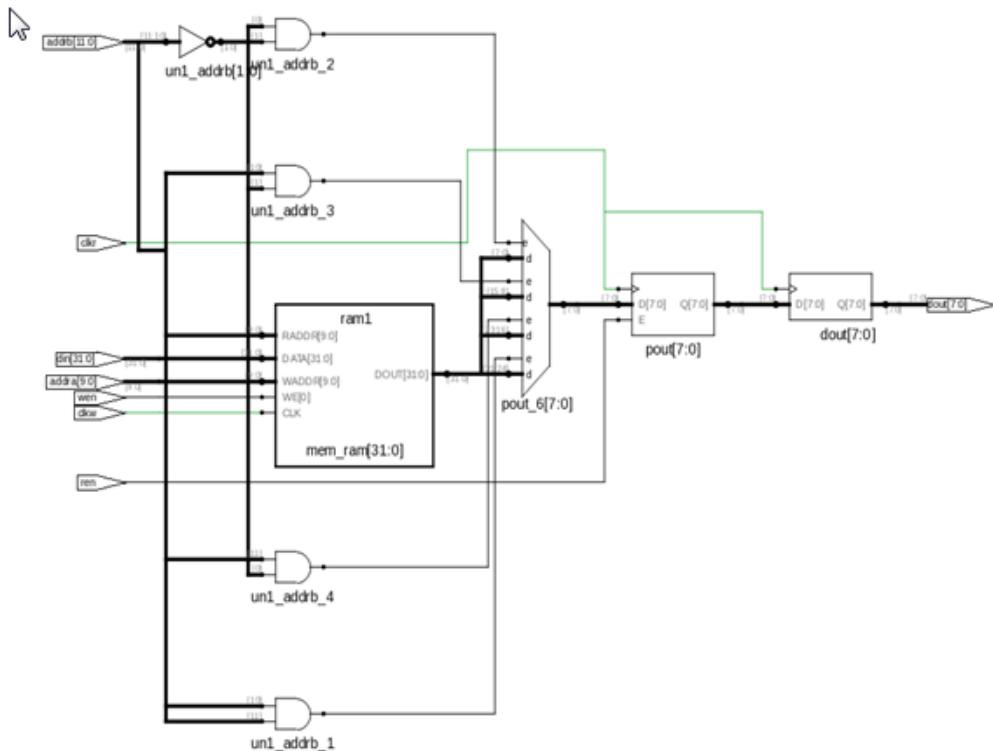
```
localparam max_width=(rwidth>wwidth)?rwidth:wwidth;
localparam min_width=(rwidth<wwidth)?rwidth:wwidth;
localparam ratio= max_width/min_width;

input [wwidth-1:0] din;
input clkw, clkr, rst, wen, ren;
input [$clog2(wdepth)-1:0] addra;
input [$clog2(rdepth)-1:0] addrb;
output reg [rwidth-1:0] dout;
reg [rwidth-1:0] pout;
reg [wwidth-1:0] temp;
wire [$clog2(ratio)-1:0] sel;
reg [$clog2(rdepth)-1:0] taddrb;
reg [max_width-1:0] mem_ram[wdepth-1:0];

always @ (posedge clkw)
begin
    if (wen)
        mem_ram[addra][max_width-1:0] <=din;// Symmetric RAM
end

assign sel=addrb[$clog2(rdepth)-1 -: $clog2(ratio)];//
always @ (posedge clkr)
begin
    taddrb<=addrb;
    dout<=pout;
end

always @ (posedge clkr)
begin
    temp=mem_ram[addrb[$clog2(wdepth)-1:0]];
    if (ren)
        case (sel)
            0: pout<=temp[rwidth-1:0];
            1: pout<=temp[2*rwidth-1:rwidth];
            2: pout<=temp[3*rwidth-1:2*rwidth];
            3: pout<=temp[4*rwidth-1:3*rwidth];
        endcase
    end
endmodule
```



1Kx32 write and 4Kx8 read configured asymmetric RAM

Resource utilization for 4Kx8 read and 1Kx32 write asymmetric RAM contains:

- BRAM 1 (read width=8, write width=32)
- DFF 8 (read address register)

Asymmetric RAM Limitations

Here are Achronix asymmetric RAM limitations:

- Currently, true dual-port asymmetric RAM is not supported.
- If an asymmetric RAM is implemented using N-parallel write access for a write width > read width configuration, then the compiler infers NRAM.

NRAM can only be mapped to registers (when `syn_max_memsize_reg > ram size`) or block RAMs. If the attribute `syn_ramstyle = "logic_ram"` is applied on the RAM, the tool generates the error message ID MF680.

Example 5: Asymmetric RAM Using NRAM and `syn_max_memsize_reg`

In this example an asymmetric RAM with 1Kx80 write and 8Kx10 read configuration is implemented using NRAM, for which `syn_max_memsize_reg` is re-constrained to 81921. The tool maps this asymmetric RAM using registers and LUT4s.

```
module aram(rst, din ,dout, addra, addrb, clkr,clkw, wen, ren)
    /*synthesis syn_max_memsize_reg=81921*/;
    parameter rwidth=10;
    parameter wwidth=80;
    parameter wdepth=1024;
    parameter rdepth=8*1024;
    localparam max_width=(rwidth>wwidth)?rwidth:wwidth;
    localparam min_width=(rwidth<wwidth)?rwidth:wwidth;
    localparam ratio= max_width/min_width;
    input rst;
    input [wwidth-1:0] din;
    input clkw, clkr, wen, ren;
    input [$clog2(wdepth)-1:0] addra;
    input [$clog2(rdepth)-1:0] addrb;
    output reg [rwidth-1:0] dout;
    reg [$clog2(rdepth)-1:0] taddrb;
    reg [min_width-1:0] mem_ram[rdepth-1:0];

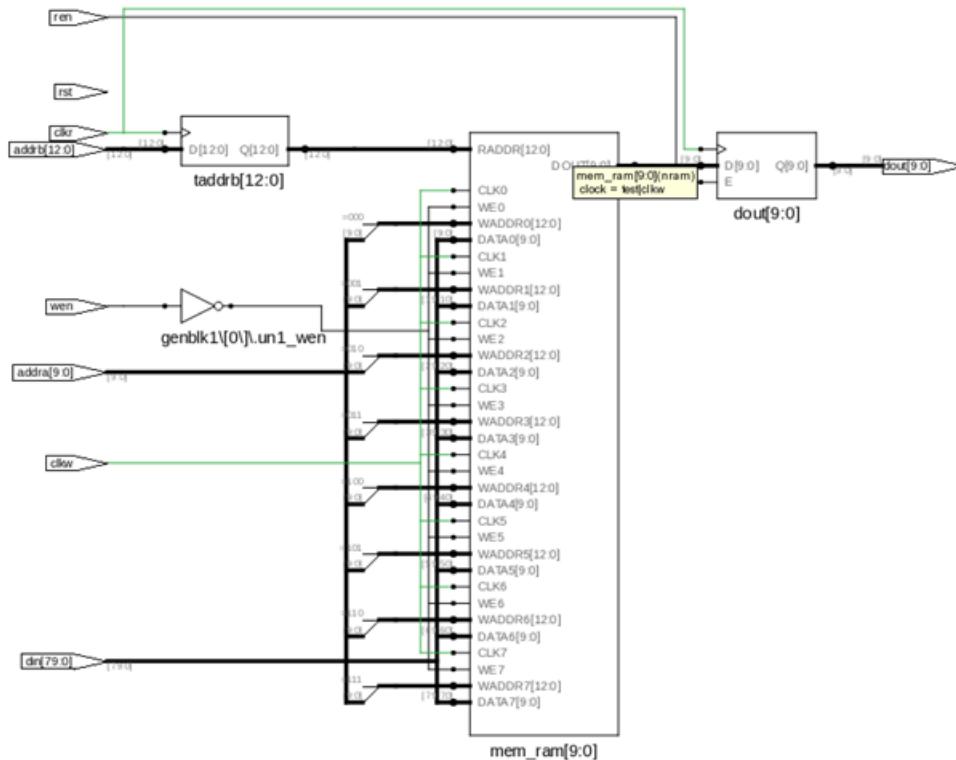
    always @ (posedge clkr)
        taddrb<=addrb;

    genvar i;
    generate
        for (i=0;i<ratio;i=i+1)
        begin
            always @ (posedge clkw)
                begin
                    if (~wen)
                        mem_ram[{i[$clog2(ratio)1:0],addra}]
                            <=din[min_width*(i)+:min_width];
                end
        end
    endgenerate
    always @ (posedge clkr)
```

```

begin
    if(ren)
        dout<=mem_ram[taddrb];
end
endmodule

```



RTL view of inferred NRAM

- **Read/Write Check**

Currently, asymmetric RAM inference is not supported when read/write check is set to on. If asymmetric RAM is implemented using NRAM, (as shown in the example above), then the RAM is mapped to registers when `syn_max_memsize_reg > ram size`.

Achronix LRAM2K_SDP Inference

The Achronix Speedster7t device supports Logic RAM (LRAM2K_SDP) 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:

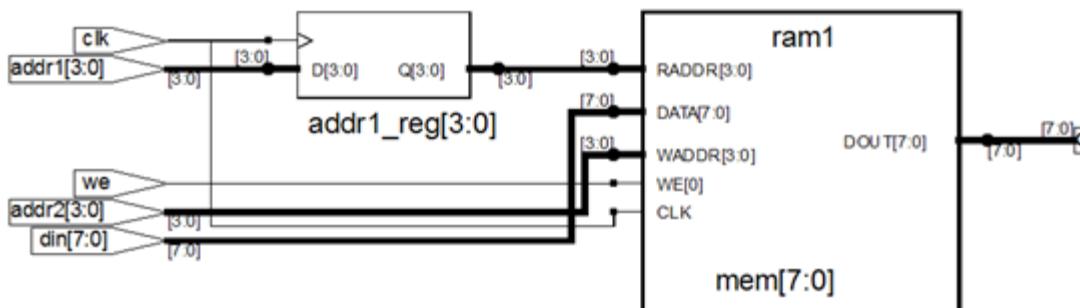
- LRAM2K_SDP is inferred if the RAM size is between 128 and 6400.
- BRAM72K_SDP is inferred if the RAM size is greater than 6400.
- Registers are inferred if the RAM size is less than 128.

For bigger RAM (for example, address width greater than 6 and/or data width greater than 10) the software can infer more than one LRAM2K_SDP.

The following combinations of LRAM2K_SDP implementations are possible:

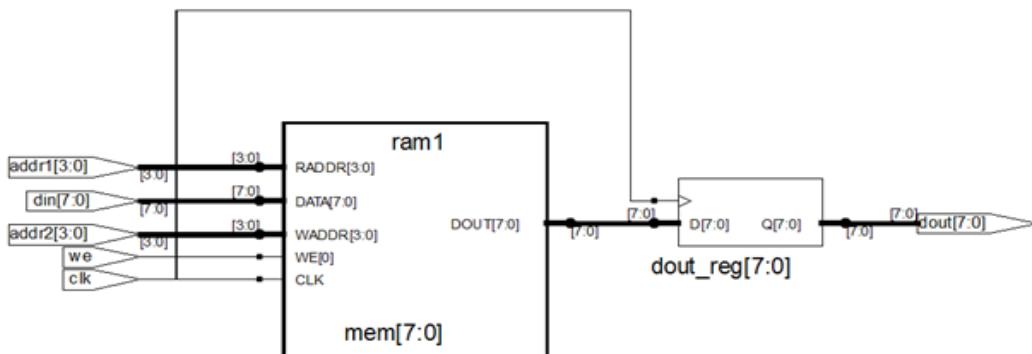
- [Example 1: RAM With Read-Address Register](#)
- [Example 2: RAM With Output Register](#)
- [Example 3: RAM with both read-address register and output register](#)
- [Example 4: Async read RAM support with different read and write addresses](#)
- [Example 5: Async read RAM support with same read and write address](#)
- [Example 6: Read Only Memory \(ROM\)](#)

Example 1: RAM With Read-Address Register



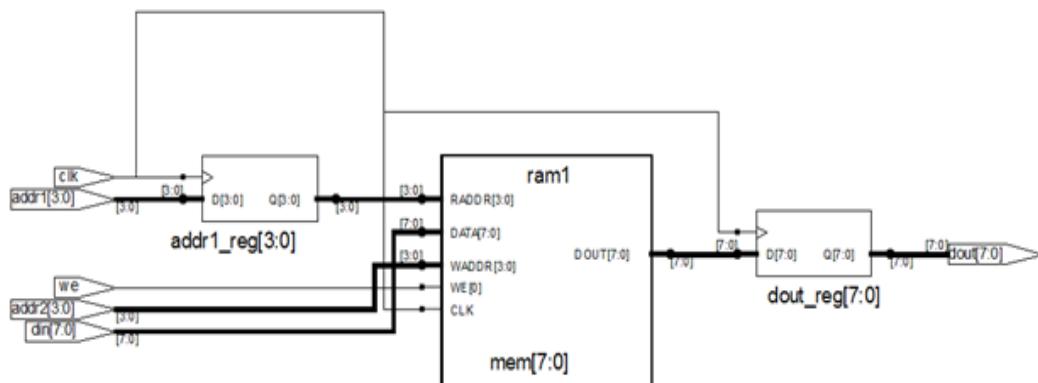
When the `syn_ramstyle` attribute is specified as `no_rw_check`, the software infers LRAM2K_SDP without glue logic for memory collision resolution. Otherwise, LRAM2K_SDP is inferred with glue logic.

Example 2: RAM With Output Register



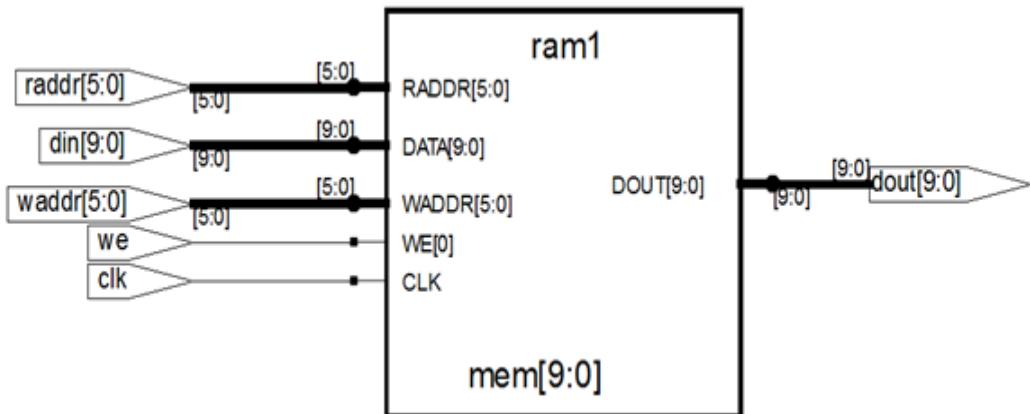
When the `syn_ramstyle` attribute is specified as `no_rw_check`, the software infers LRAM2K_SDP without glue logic for memory collision resolution. Otherwise, LRAM is not inferred and RAM is mapped to registers.

Example 3: RAM with both read-address register and output register

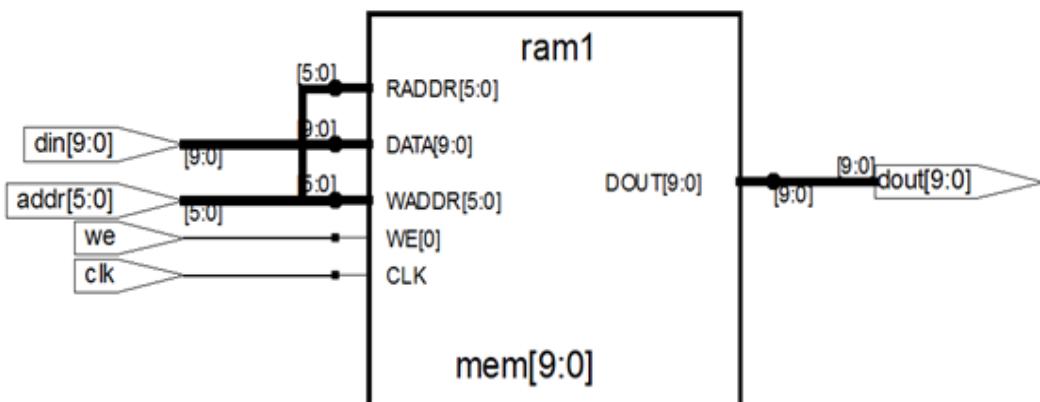


When the `syn_ramstyle` attribute is specified as `no_rw_check`, the software infers LRAM with output register packed and without glue logic for memory collision resolution. Otherwise, LRAM is inferred with input read-address register packed and with glue logic for memory collision resolution.

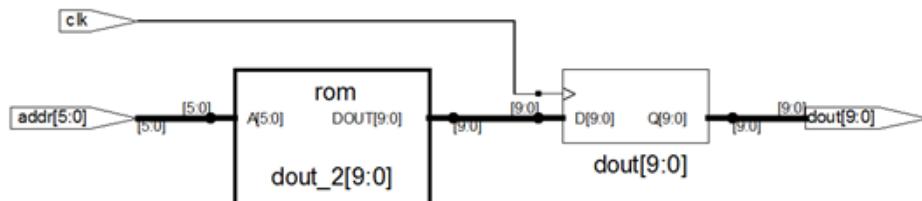
Example 4: Async read RAM support with different read and write addresses



Example 5: Async read RAM support with same read and write address

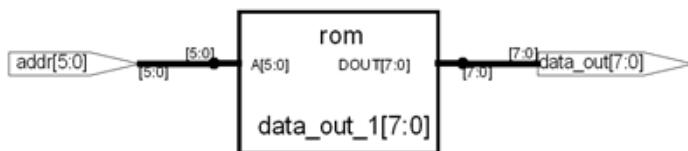


Example 6: Read Only Memory (ROM)



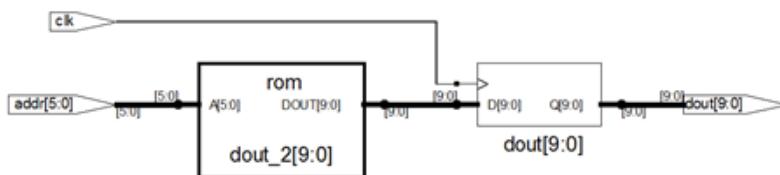
If the ROM size is between 128 and 6400, it is mapped to LRAM2K_SDP.

Asynchronous ROM



Asynchronous ROMs of size greater than 64x8 (ROM depth greater than 64 and ROM data width greater than 8) are mapped to LRAM2K_SDP.

Synchronous ROM



Synchronous ROMs of size greater than 64x8 and less than 6400 are mapped to LRAM2K_SDP.

Memory Initialization

The initial values specified using \$readmemb and \$readmemh statements are forward annotated as mem_init parameter value of LRAM2K_SDP.

Limitations

Limitations include the following:

- True dual-port RAM cannot be mapped to LRAM2K_SDP.
- If the read address register is initialized, then it does not get packed in the LRAM2K_SDP.
- Parameter mem_init_file is not supported. However, if initial values are given by data_file in the RTL, then those initial values are applied to LRAM2K_SDP using the parameter mem_init.

Packing Output Registers in the LRAM2K_SDP

Speedster7t Technology

The tool packs output registers into the LRAM2K_SDP when their reset pins have priority over the enable pins.

In the following example, the tool packs the output register in the LRAM2K_SDP:

Example: Packing Output Registers in the LRAM2K_SDP

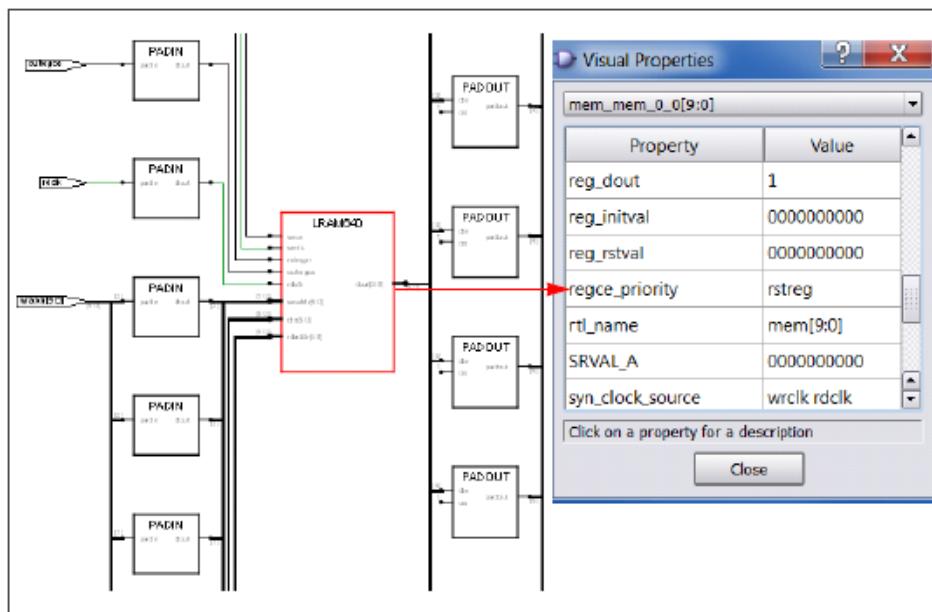
```
module lram_infer_reset_priority_over_enable
    (input [9:0] din, input [5:0] wraddr, rdaddr, input wrclk,
     wren, outregce, outregrstn, rdclk, output [9:0] dout);
    reg [9:0] mem [63:0];
    reg [9:0] dout_r;
    always @ (posedge wrclk)
        if (wren)
            mem[wraddr] <= din;
    always @ (posedge rdclk)
```

```

if (~outregrstn)
    dout_r <= '0;
else if (outregce)
    dout_r <= mem[rdaddr];
assign dout = dout_r;
endmodule

```

The regce_priority property is set to rstreg for the LRAM2K_SDP in the Technology view.



Achronix BRAM72K_SDP Inference

Achronix Speedster7t technologies support block RAM (BRAM72K_SDP) inference. BRAM72K_SDP is an 80K-bit dual-port memory.

BRAM Mapping

The BRAM72K_SDP 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 BRAM72K_SDP. 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 BRAM72K_SDP 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_ram-style` can be used.

The synthesis tool supports the following BRAM72K_SDP memory configurations:

- True dual-port RAM
- Simple dual-port RAM
- Single-port RAM
- Read only memory (ROM)

BRAM72K_SDP is inferred irrespective of whether the output registers are uninitialized or initialized in both the latched and registered modes.

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.

ROM Optimization

The tool performs ROM optimizations based on initial values defined in the memory initialization file. For example:

- When the Verilog readmemb and readmemh constructs are used to provide initial values, by default, the tool does not perform ROM optimization.
- The tool does perform ROM optimization if the:
 - Coding style uses Verilog or VHDL case statements.
 - Constant array of parameters in Verilog is followed by a MUX.
 - Initial values are specified in the RTL source code.

The `syn_preserve_rom` directive overrides the default optimization behavior. The directive can only be applied in the RTL source code.

ROM Optimization Example 1

The memory is initialized using `readmemb` in Verilog and the tool does not optimize the initial values for this example.

```
module test (addr, clk , data_out);
`else
module test_RTL (addr, clk , data_out);
`endif

parameter ADDR_WIDTH = 4;
parameter DATA_WIDTH = 8;
parameter MEM_DEPTH = 16;

input [ADDR_WIDTH-1:0]addr;
input clk;
output reg [DATA_WIDTH-1 : 0]data_out;
reg [DATA_WIDTH-1 : 0]mem[MEM_DEPTH-1 : 0];

initial
begin
    $readmemb ("data.dat",mem);
end

always @ (posedge clk)
    data_out <= mem[addr];

endmodule
```

The contents of the data.dat file:

```
00001111
00000101
00000101
00001111
00000101
00000101
00001111
00000101
00000101
00001111
00000101
00000101
00001111
00000101
00000101
00001111
00000101
00000101
00001111
00000101
00000101
00001111
```

In this case, the tool does not perform ROM optimization.

ROM Optimization Example 2

In this example, the ROM is defined using the `case` statement in VHDL and the tool performs optimization to reduce the size of the RAM.

```
library ieee;
use ieee.std_logic_1164.all;

entity ROM is
    port (clk :in std_logic;
          address : in std_logic_vector(3 downto 0);
          data_out : out std_logic_vector(7 downto 0));
end entity ROM;

architecture behavioral of ROM is
    type mem is array ( 0 to 2**4 - 1)
        of std_logic_vector(7 downto 0);
    signal data: std_logic_vector(7 downto 0);
    constant my_Rom : mem := (
        0 => "00000000",
        1 => "00000000",
        2 => "00000000",
        3 => "00000000",
        4 => "00000100",
```

```
5  => "11110000",
6  => "11110000",
7  => "11110000",
8  => "11110000",
9  => "11110000",
10 => "11110000",
11 => "11110000",
12 => "11110000",
13 => "11110000",
14 => "11110000",
15 => "11110000");

begin
process (address)
begin
    case address is
        when "0000" => data <= my_rom(0);
        when "0001" => data <= my_rom(1);
        when "0010" => data <= my_rom(2);
        when "0011" => data <= my_rom(3);
        when "0100" => data <= my_rom(4);
        when "0101" => data <= my_rom(5);
        when "0110" => data <= my_rom(6);
        when "0111" => data <= my_rom(7);
        when "1000" => data <= my_rom(8);
        when "1001" => data <= my_rom(9);
        when "1010" => data <= my_rom(10);
        when "1011" => data <= my_rom(11);
        when "1100" => data <= my_rom(12);
        when "1101" => data <= my_rom(13);
        when "1110" => data <= my_rom(14);
        when "1111" => data <= my_rom(15);
        when others => data <= "00000000";
    end case;
end process;

process(clk)
begin
    if rising_edge(clk) then
        data_out <= data;
    end if;
end process;
end architecture behavioral;
```

In this case, the tool optimizes away bits 0, 1, and 3 of the ROM; where these bits are defined as 0 in the RTL.

RAM and ROM Initialization for Achronix

To initialize RAM and ROM, see [Initializing Values in RTL, on page 495](#).

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 528 in the *User Guide* for step-by-step procedures. You can initialize values in Verilog and VHDL. See:

- [Initializing RAMs in Verilog](#), on page 528
- [Initializing RAMs in VHDL](#), on page 529

Achronix MLP primitives Inference

The tool infers MLP primitives for Speedster7t devices. The following logic structures can be mapped to MLP primitives:

- 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 MLP primitives 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 MLP primitives.

The input/output registers for the multiplier, mult-add, mult-sub can be packed into the MLP primitives. Register packing is implemented even when the registers cross hierarchy.

- Mapping both signed and unsigned multipliers to MLP primitives.
- Multipliers with a size greater than 28X28 (signed) and 27X27 (unsigned) are fractured and mapped to multiple MLP primitives.

Mult-add/Mult-sub Inference

The tool infers MLP primitives for Mult-add and Mult-sub by using the cascade chain. Prerequisites for inferring MLP primitives 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 MLP primitives block and there are no external feedback paths. Prerequisites for inferring MLP primitives 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 MLP primitives. The MLP primitives is configured so that it can load the multiplier output.

Example: RTL for Loadable Accumulator Mapped to MLP primitives

```

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 Latch Inference

The Achronix Speedster7t device supports inference of latches. Achronix implements the latch primitives shown in the table below for inferencing:

Latch Primitive	Description
LAT	Latch without any control signals
LATN	Active low level sensitive latch
LATE	Latch with enable
LATNE	Active low level sensitive latch with enable
LATER (with sr_assertion = "unclocked")	Latch with asynchronous reset and enable. Reset has priority over enable.

Latch Primitive	Description
LATNER (with sr_assertion = "unclocked")	Active low level sensitive latch with asynchronous reset and enable. Reset has priority over enable.
LATES (with sr_assertion = "unclocked")	Latch with asynchronous set and enable. Set has priority over enable.
LATNES (with sr_assertion = "unclocked")	Active low level sensitive latch with asynchronous set and enable. Set has priority over enable.
LATR (with sr_assertion = "unclocked")	Latch with asynchronous reset
LATNR (with sr_assertion = "unclocked")	Active low level sensitive latch with asynchronous reset
LATS (with sr_assertion = "unclocked")	Latch with asynchronous set
LATNS (with sr_assertion = "unclocked")	Active low level sensitive latch with asynchronous set

Note: When a latch has the following:

- An enable and synchronous set/reset with the enable having higher priority over the reset/set, the LATE primitive is inferred with the reset/set signal logic mapped to LUT4.
- An enable and synchronous set/reset with the reset/set having higher priority over the enable, the LATE[RS] primitive is inferred with the parameter sr_assertion set to unclocked.
- Both asynchronous set and reset, the tool generates an error.

Here are several examples of latch inferencing for Achronix.

Example 1: Latch with no control signal

In this example, the latch does not have control signals.

Verilog Example1

```
//TECHPUBS Verilog Example 1: Latch with no control signal
```

```
module test (d, ck, q);  
    input d, ck;  
    output q;  
    reg q;  
  
    always @ (ck or d)  
        if (ck)  
            q = d;  
  
endmodule
```

VHDL Example 1

--TECHPUBS VHDL Example 1: Latch with no control signal

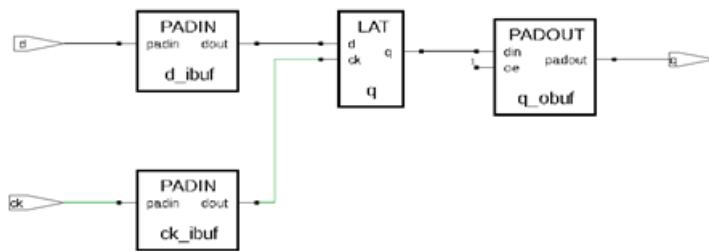
```
library ieee;  
use ieee.std_logic_1164.all;  
entity test is port  
(  
    ck : in std_logic;  
    d : in std_logic;  
    q: out std_logic);  
end entity test;  
architecture rtl of test is  
begin  
process(ck,d)  
begin
```

```

if ck='1' then
    q <= d;
end if;
end process;
end rtl;

```

With the above code, the tool implements the LAT primitive in the netlist:



Example 2: Latch with enable

This example specifies the latch with an enable input.

Verilog Example 2

```
//TECHPUBS Verilog Example 2: Latch with enable
```

```

module test (d, en, ck, q);
    input d,en,ck;
    output q;
    reg q;

    always @ (ck or d or en)
        if (ck && en)
            q = d;

```

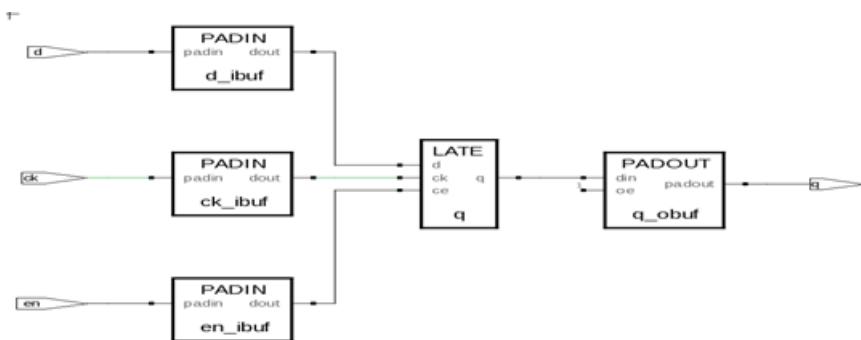
```
endmodule
```

VHDL Example 2

```
--TECHPUBS VHDL Example 2: Latch with enable

library ieee;
use ieee.std_logic_1164.all;
entity test is port
(
    ck,en : in std_logic;
    d : in std_logic;
    q: out std_logic);
end entity test;
architecture rtl of test is
begin
process(ck,d,en)
begin
    if ck='1' then
        if en = '1' then
            q <= d;
        end if;
    end if;
end process;
end rtl;
```

The tool implements the LATE primitive as shown below:



Example 3: Active low level sensitive latch with asynchronous reset

In this example, the latch is active low level sensitive with an asynchronous reset signal.

Verilog Example 3

```
//TECHPUBS Verilog Example 3: Active low level sensitive latch with
asynchronous reset
```

```
module test ( input d, ck, rn, output reg q );
    always @ ( d or rn or ck )
        if ( !rn )
            q <= 1'b0;
        else
            if ( !ck )
                q <= d;
endmodule
```

VHDL Example 3

```
--TECHPUBS VHDL Example 3: Active low level sensitive latch with
asynchronous reset
```

```
library ieee;
use ieee.std_logic_1164.all;
entity test is port
(
    ck,rn : in std_logic;
    d : in std_logic;
    q: out std_logic);
```

```

end entity test;

architecture rtl of test is

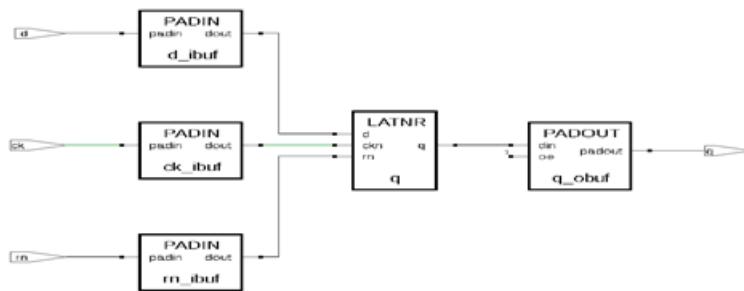
begin

process(ck,d,rn)
begin

if rn = '0' then
    q <= '0';
elsif ck='0' then
    q <= d;
end if;
end process;
end rtl;

```

With this code, the tool infers the LATNR primitive with sr_assertion="unclocked" in the netlist.



Example 4: Latch with asynchronous set

The example below implements a latch with asynchronous set signal.

Verilog Example 4

```
//TECHPUBS Verilog Example 4: Latch with asynchronous set
```

```
module test ( input d, ck, sn, output reg q );
    always @ ( d or sn or ck )
        if ( ! sn )
            q <= 1'b1;
        else
            if ( ck )
                q <= d;
endmodule
```

VHDL Example 4

--TECHPUBS VHDL Example 4: Latch with asynchronous set

```
library ieee;
use ieee.std_logic_1164.all;
entity test is port
(
    ck,sn : in std_logic;
    d : in std_logic;
    q: out std_logic);
end entity test;
architecture rtl of test is
begin
process(ck,d,sn)
begin
    if sn = '0' then
```

```

q <= '1';

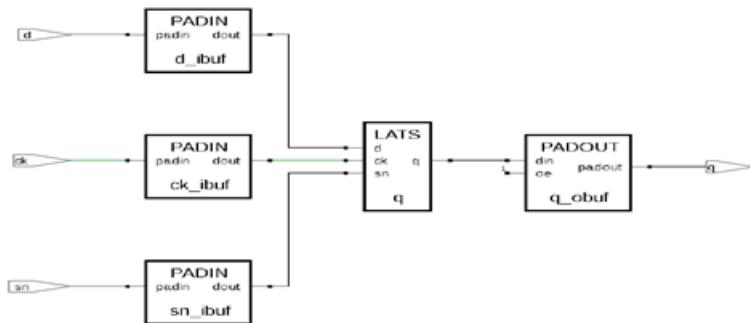
elsif ck='1' then
    q <= d;
end if;

end process;

end rtl;

```

The tool infers LATS primitive with sr_assertion="unclocked" for this code as shown below:



Example 5: Latch with enable and synchronous set and enable having priority

In this example, the latch has an enable and a synchronous set. The enable has higher priority than the set signal.

Verilog Example 5

```
//TECHPUBS Verilog Example 5: Latch with enable and synchronous set
and enable having higher priority
```

```

module test ( input d, ck, sn,en,output reg q );
    always @ ( d or sn or en or ck )

```

```
if (ck && en)
    if ( ! sn )
        q <= 1'b1;
    else
        q <= d;
endmodule
```

VHDL Example 5

--TECHPUBS VHDL Example 5: Latch with enable and synchronous set and enable having higher priority

```
library ieee;
use ieee.std_logic_1164.all;
entity test is port
(
    ck,sn,en : in std_logic;
    d : in std_logic;
    q: out std_logic);
end entity test;
architecture rtl of test is
begin
process(ck,d,sn,en)
begin
    if (ck = '1') then
        if (en = '1') then
            if sn = '0' then
                q <= '1';
```

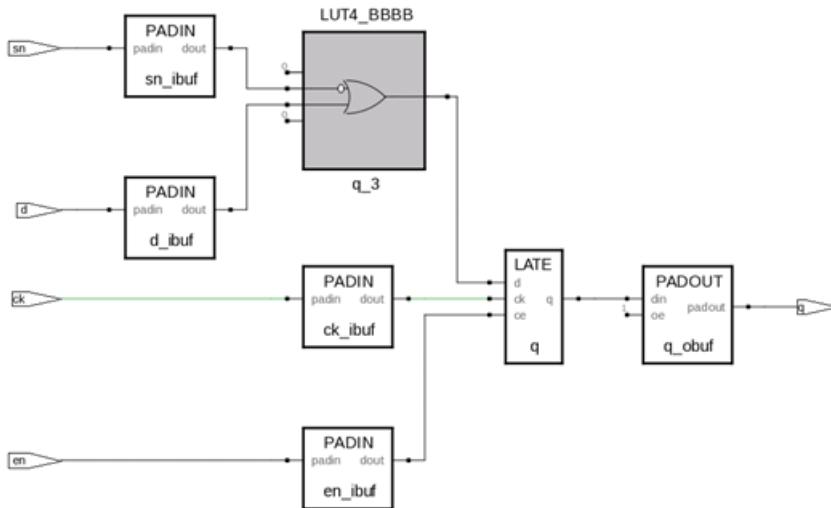
```

        else
            q <= d;
        end if;
    end if;
    end if;

end process;
end rtl;

```

For the code above, the tool implements LATE with extra logic for the set signal being mapped to LUT4s as shown below:



Example 6: Active low level sensitive latch with enable and synchronous reset with reset having priority over enable

In this example, an active low level sensitive latch has an enable and a synchronous reset. The reset signal has higher priority than the enable.

Verilog Example 6

//TECHPUBS Verilog Example 6: Active low level sensitive latch with enable and synchronous reset with reset having priority over enable

```
module test ( input d, ck, rn,en,output reg q );

    always @ ( d or rn or en or ck )

        if (!ck)

            if ( ! rn )

                q <= 1'b0;

            else if (en)

                q <= d;

    endmodule
```

VHDL Example 6

--TECHPUBS VHDL Example 6: Active low level sensitive latch with enable and synchronous reset with reset having priority over enable

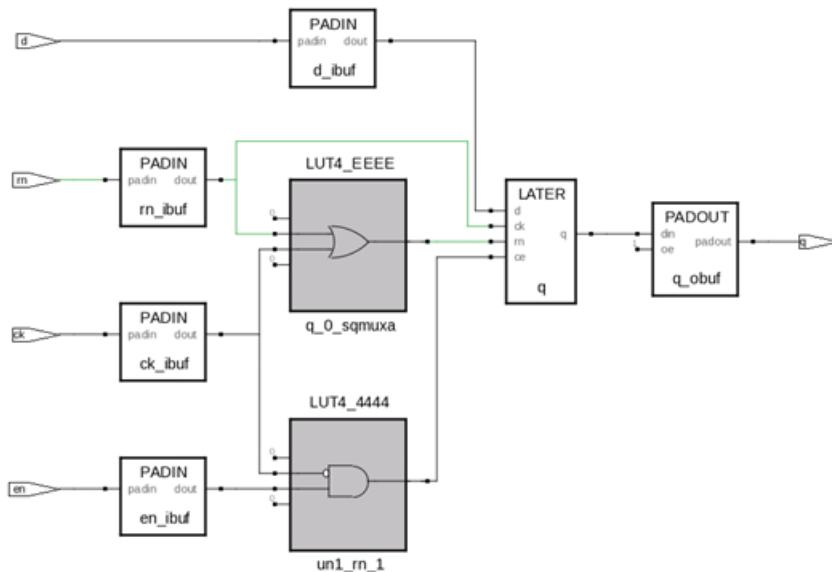
```
library ieee;
use ieee.std_logic_1164.all;
entity test is port
(
    ck,rn,en : in std_logic;
    d : in std_logic;
    q: out std_logic);
end entity test;
architecture rtl of test is
begin
process(ck,d,rn,en)
```

```

begin
  if (ck = '0') then
    if rn = '0' then
      q <= '0';
    elsif (en = '1') then
      q <= d;
    end if;
  end if;
end process;
end rtl;

```

For this code, the tool infers LATER primitive with sr_assertion="unclocked" and the extra logic is mapped to LUT4s.



Limitation

When there is an initial value specified at the output of the latch, the tool displays an error.

Achronix Register Inference

The synthesis tool supports the following register inference features for the Achronix devices:

- Register initialization that supports all Achronix register primitives.
- 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 ¹	
dffr ²	dffc
dffer ³ (sr_assertion=unclocked)	dffer (sr_assertion=clocked)
dffs ⁴	dffp
dfxes ⁵ (sr_assertion=unclocked)	dffes (sr_assertion=clocked)
	dffec, dffep ⁶

1. dffe is a clock enabled flip-flop.

2. dffs/dffr is a flip-flop with asynchronous set/reset signal.

3. dffer with a clock sr_assertion value is an enabled flip-flop with asynchronous reset.

4. dffs/dffr is a flip-flop with asynchronous set/reset signal.

5. dfxes with a clock sr_assertion value is an enabled flip-flop with synchronous set.

6. dffec/dffep is a flip-flop with synchronous set/reset control signal.

Register Initialization for Achronix

Speedster7t Technology

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.

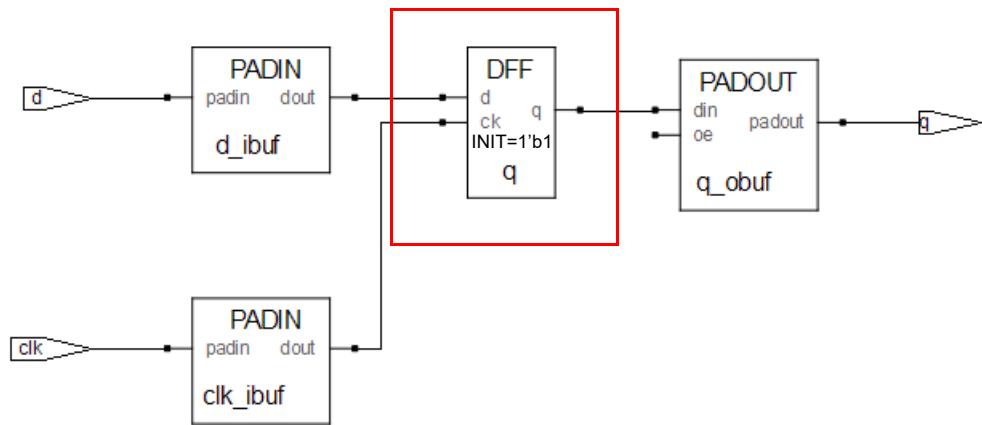
See the following examples.

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.

Case 2 - Register with Enable

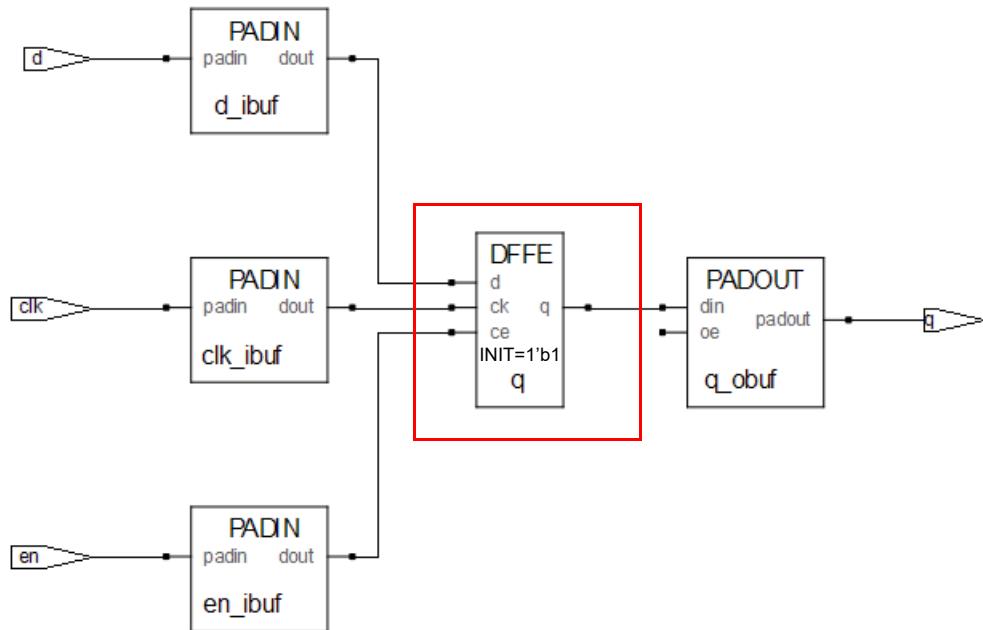
In this example, the register includes an enable with initial value = 1'b1;

```

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

```



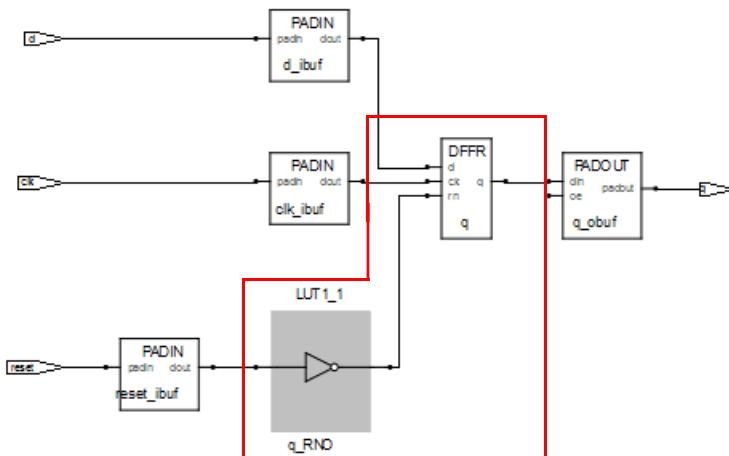
When the register initial value = 1'b0, the same RTL is generated except the DFFE does not include an init value.

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
```



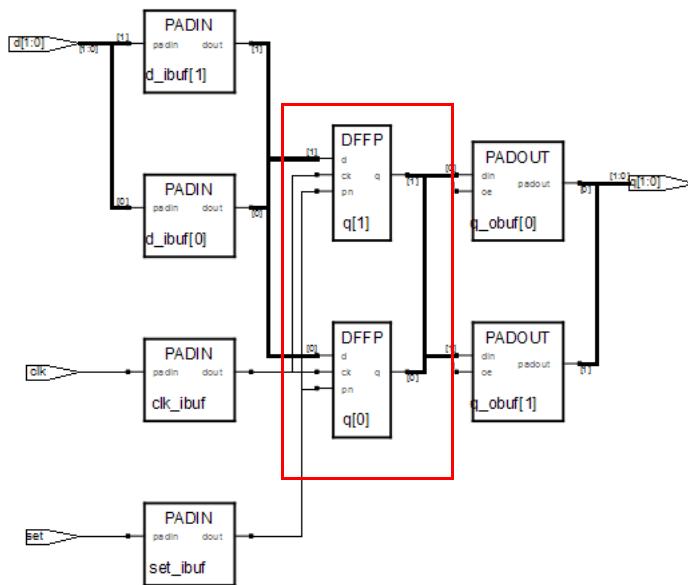
Note: Clock and control signals (reset/set/enable) with -ve polarity are also supported.

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
```

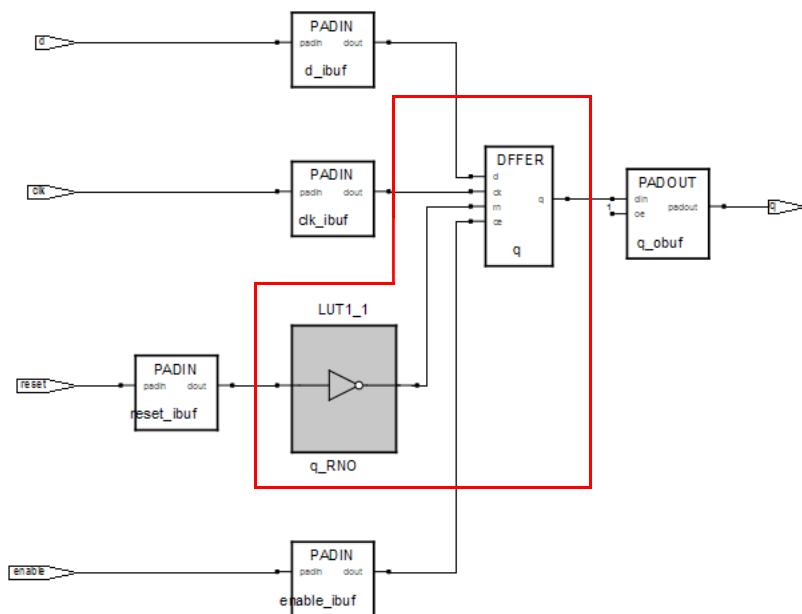


Case 5 - Register with Asynchronous Control Signals and an Enable

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;

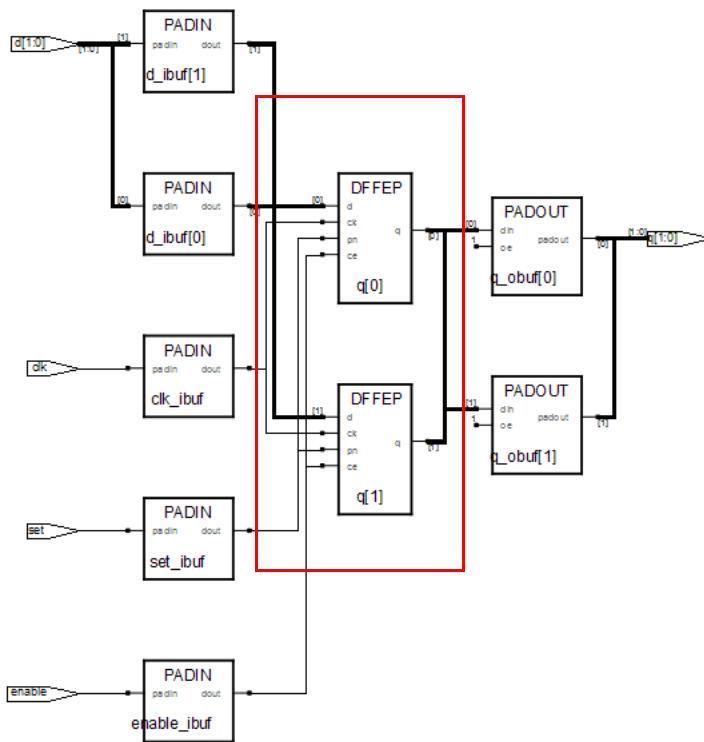
```
module test (clk, set, enable, d, q);
    input clk;
    input set;
    input enable;
    input [1:0] d;
    output [1:0] q;
    reg [1:0] q=2'b00;

    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.

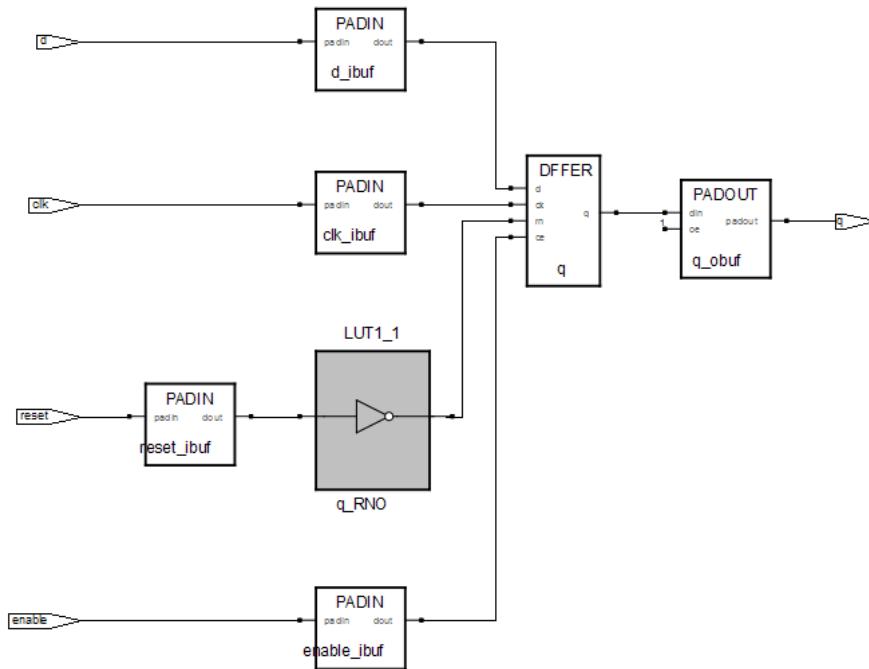
Case 1 - Register with asynchronous control signals

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
```



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.

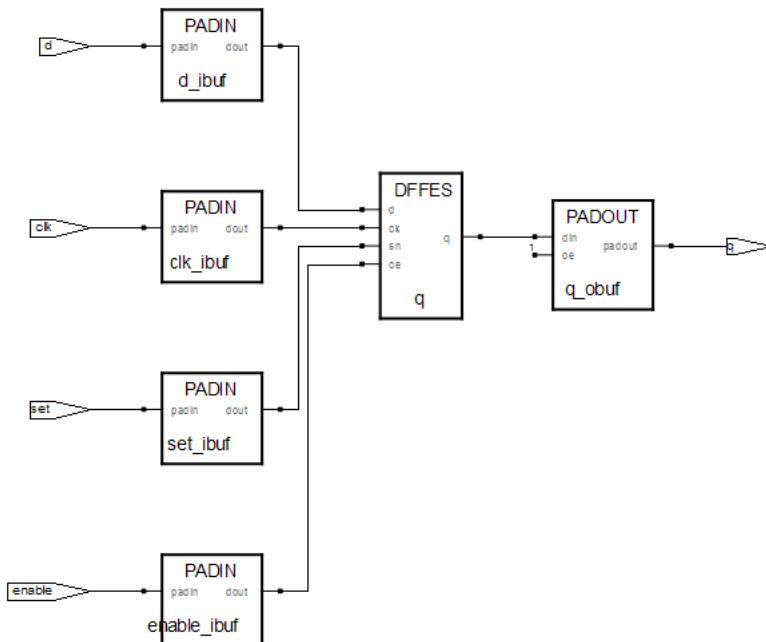
```

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

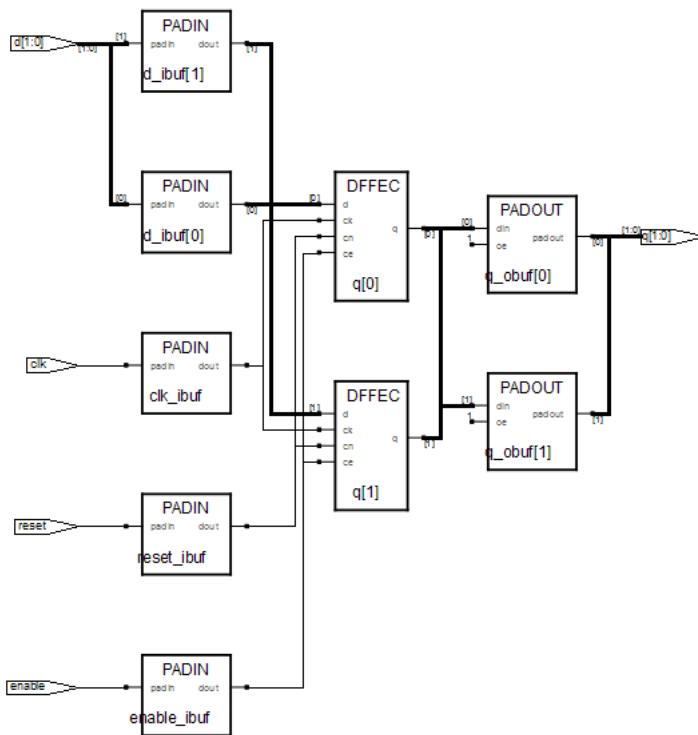
Case 3 - Register with clock enable

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.

Achronix Shift Register Inference

The software infers shift registers mapped to LRAM2K_SDP, based on threshold limits for the Achronix architecture. The RTL design can only infer the sequential shift component if the following conditions are met:

- All registers are of the same type and use the same control signals.

- The width and depth of the shift register must be greater than the threshold limits. Current threshold limits for inferring the seqshift component using LRAM2K_SDP include the following:
 - DEPTH_THRESHOLD (SRL_DEPTH) = 4
 - WIDTH_THRESHOLD (SRL_WIDTH) = 6
 - MEM_THRESHOLD (SRL_WIDTH * SRL_DEPTH) = 40
 - LRAM2K_SDP_DEPTH = 64

The seqshift maps to LRAM2K_SDP when SRL_DEPTH, SRL_WIDTH, and SRL_WIDTH * SRL_DEPTH are greater than or equal to the corresponding threshold limits. Otherwise, the seqshift is mapped to registers.

- If the depth of seqshift is greater than LRAM2K_SDP_DEPTH, then it gets divided into a chain of smaller seqshift with depth equal to or less than the LRAM2K_SDP_DEPTH. The seqshift components are mapped to LRAM2K_SDP, depending on the threshold limits.
- The seqshift is inferred for synchronous set/reset or asynchronous set/reset registers. Extra glue logic is created for the synchronous or asynchronous set/reset signals.

Use the `syn_srlstyle` attribute to change the automatic mapping of shift registers, as specified by the value set for the attribute. You can set this attribute globally or on individual registers.

Limitations

Shift registers are not inferred for the following conditions:

- The seqshift for LRAM2K_SDP inference is not supported if the output comes from a dynamic stage. This means that the RADDR input of the seqshift must be constant to infer LRAM2K_SDP. Otherwise, it is implemented using registers.
- LRAM2K_SDP cannot infer registers that have both synchronous and asynchronous reset or synchronous and asynchronous set.
- If sequential shift registers have both reset and set control signals, then registers are inferred.
- If sequential shift registers with enable signals have higher priority than synchronous set/reset signals, then registers are inferred.

Shift Register (VHDL Example)

In this VHDL code example, the seqshift maps to a chain of LRAM2K_SDP components in the HDL Analyst Technology view.

```

library ieee;
use ieee.std_logic_1164.all;

entity p_seqshift_rtl is
    generic (SRL_WIDTH : natural := 12;
             SRL_DEPTH : natural := 1026);
    port(clk : in std_logic;
         din : in std_logic_vector((SRL_WIDTH-1) downto 0);
         dout : out std_logic_vector((SRL_WIDTH-1) downto 0));
end p_seqshift_rtl;
architecture archi of p_seqshift_rtl is
type shift is array ((SRL_DEPTH-1) downto 0) of
    std_logic_vector((SRL_WIDTH-1) downto 0);
    signal tmp: shift;
begin
begin
    process (clk)
        begin
            if rising_edge(clk) then
                for i in (SRL_DEPTH-1) downto 1 loop
                    tmp(i) <= tmp(i-1);
                end loop;
                tmp(0) <= din;
            end if;
        end process;
        dout <= tmp(SRL_DEPTH-1);
    end archi;

```

Shift Register (Verilog Example)

In this Verilog code example, the seqshift maps to a chain of LRAM2K_SDP components in the HDL Analyst Technology view as well.

```

module p_seqshift (clk, din, dout);

parameter SRL_WIDTH = 12;
parameter SRL_DEPTH = 1026;

input clk;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0];

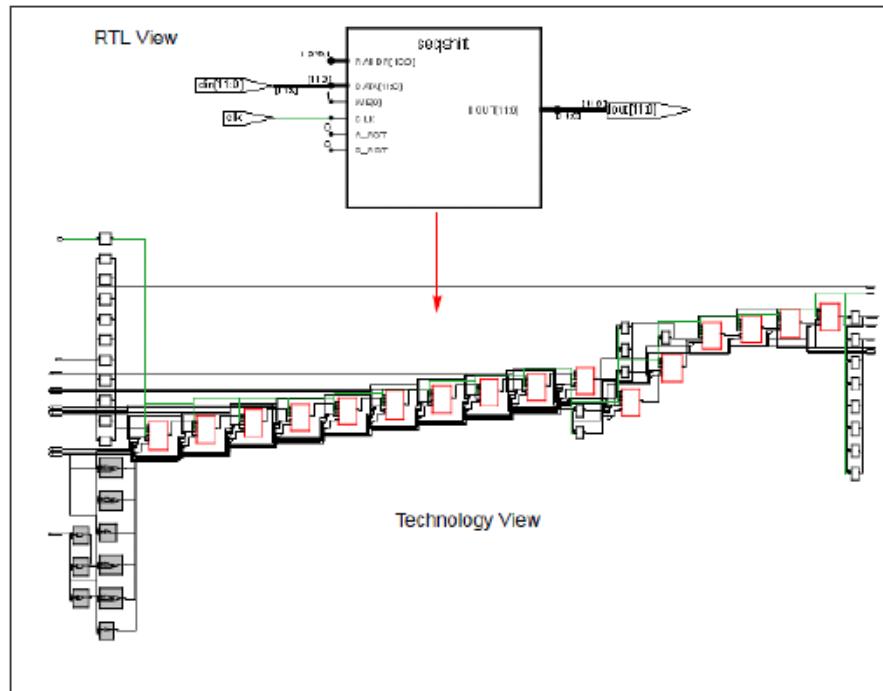
```

```
integer i;

always @ (posedge clk)
begin
    for (i=SRL_DEPTH-1; i>0; i=i-1)
    begin
        regBank[i] <= regBank[i-1];
    end
    regBank[0] <= din;
end
assign dout = regBank[SRL_DEPTH-1];

endmodule
```

The following figure shows the seqshift component in HDL Analyst RTL view mapped to a chain of LRAM2K_SD_P components in the Technology view.



Achronix Device Mapping Options

To achieve optimal design results, set the correct implementation options. Some options include the following:

- [Disable I/O Insertion](#), on page 527
- [Time Borrowing](#), on page 527
- [Compile Point Remapping in Achronix Designs](#), on page 528

See Also

- [Achronix set_option Command Options](#), on page 529
- [Achronix Tcl set_option Command](#), on page 531
- [Tcl project Command](#), on page 533

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 622](#) and [Compile Point Types, on page 610](#) 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.

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 <i>not</i> 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 <i>interface</i> 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 <i>updated</i> 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: <ul style="list-style-type: none">• The compile point is remapped, and the Update Compile Point Timing Data option is enabled.• The interface of the compile point is changed.

Continue on Error Mode for Compile Points

By default, the tool stops the synthesis process when an error is encountered within a compile point. If you enable Continue on Error, this option allows compile point synthesis to continue for other compile points.

With Continue on Error mode enabled, when the tool encounters a compile point mapper error, a black box is created for the affected compile point and the software continues to synthesize other compile points. The log file generates a warning for the compile point being ignored during synthesis as shown below:

```
@W: m1.v(1) | Mapping of compile point m1 - Unsuccessful
```

The Continue on Error setting applies only to compile point mapper errors. All compiler errors must be corrected before synthesis can proceed.

Automatic Compile Point (ACP) Support

The tool automatically identifies compile points based on various parameters, such as size of the design and hierarchical modules, and considering their boundary logic. You can synthesize the compile points in parallel using multiprocessing to improve runtime. For more information, see [Automatic and Manual Compile Points, on page 608](#).

Achronix set_option Command 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 447](#) and [Constraints Panel, on page 450](#).

The following table lists device mapping options (Device tab) for the Speedster7t technology.

Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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 <code>Tcl Find</code> command. See Annotating Timing Information in the Schematic Views, on page 465 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 Disable I/O Insertion, on page 527 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 577 and Controlling Buffering and Replication, on page 579 in the User Guide.
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 combinational 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 527 for more information. For the Speedster7t 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 619 , for details.

You can also use the corresponding `set_option` Tcl command to enable/disable these options. See [Achronix Tcl set_option Command](#), on page 531 for more information.

Achronix 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 Speedster7t technology. See [set_option, on page 149](#) 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: Speedster7t.
-part partName	Specifies a part for the implementation. Refer to the Implementation Options dialog box for available choices (Implementation Options Command, on page 444).
-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 444).
-package packageName	Specifies the package for the implementation. Refer to the Implementation Options dialog box for available choices (Implementation Options Command, on page 444).
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 527 for a description.
-fanout_limit value	Sets the fanout limit. Default is 1000. See Setting Fanout Limits , on page 577 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 the Design , on page 560 of the <i>User Guide</i> for information on how to use it.

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.
-resource_sharing 1 0	Enables/disables resource sharing. See Sharing Resources , on page 581 in the <i>User Guide</i> for information about using it.
-retime_registers_forward 1 0	Forward retiming registers across combinational logic towards the output to improve area QoR.
-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 563 of the <i>User Guide</i> for information on how to use it.
-rw_check_on_ram 1 0	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the syn_ramstyle attribute, see syn_ramstyle, on page 509 .
-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 585 in the <i>User Guide</i> 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 527 for more information. For the Speedster7t 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 528, for details. <ul style="list-style-type: none"> • 1 Forces remapping of parents • 0 Does not remap parents

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 Speedster7t. See [project, on page 111](#) 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 Output Files and Forward Annotation

The following describes files that forward annotate information for the Achronix back-end tools.

- [Automatic RTL Attribute Propagation](#), on page 534
- [Net Attributes in RTL](#), on page 534
- [Forward Annotation of RTL Attributes to the Netlist](#), on page 537
- [Forward Annotation of RTL Attributes Applied on Pins](#), on page 538

Automatic RTL Attribute Propagation

RTL attribute propagation automatically forward annotates user attributes without specifically using attributes like `syn_noprune` or `syn_keep`, as long as the object is preserved during synthesis. There is no guarantee that user RTL attribute propagation happens, if the object is optimized away during synthesis.

To ensure that objects are preserved with the user attributes applied, you can use them along with synthesis directives like `syn_noprune` or `syn_keep`, as described in the *Attribute Reference* manual.

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;

```

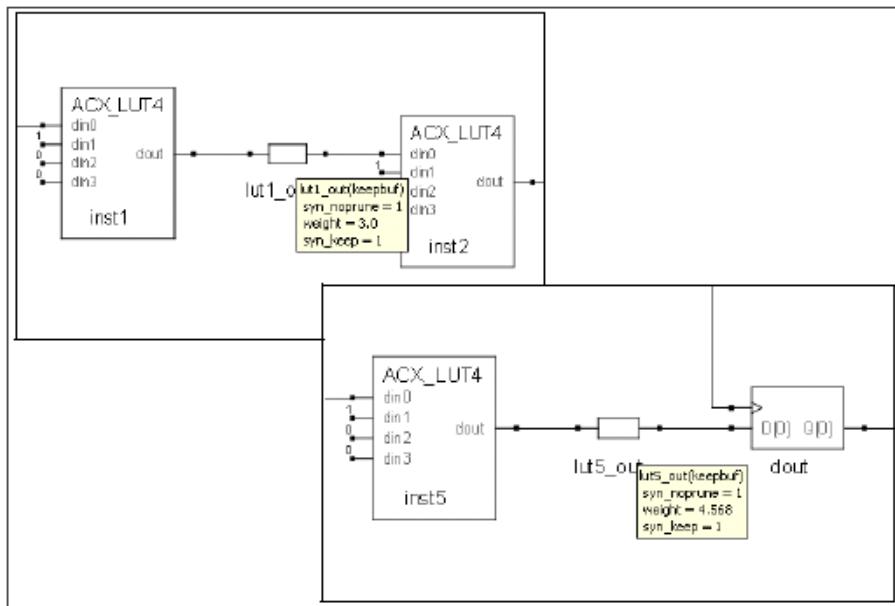
```

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.



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	<code>syn_hier="hard"</code>
Instantiated components	<code>syn_noprune</code>
Input/output ports	<code>syn_hier="hard"</code> on the module containing the ports
Registers	<code>syn_preserve</code>

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

Note: Using the synthesis directives/attributes along with the RTL attributes can prevent certain optimizations; this can impact the QoR.

See [How Attributes and Directives are Specified](#), on page 10 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.

Forward Annotation of RTL Attributes Applied on Pins

The software honors user-specific attributes applied on the pins of instances in the Verilog RTL. The synthesis tool forward-annotates these attributes with their corresponding values to the Verilog netlist (VM) file.

To forward-annotate the attributes applied on the pins, ensure that the pins of the instantiated module are not optimized and the pin names do not change. You can apply the `syn_hier="hard"` attribute on the corresponding module to avoid optimizing the pins.

Verilog Syntax and Example

`instName (.pinName(connection), (* attribute = value) .pinName(connection));`

The following example shows how to specify RTL attributes on pins:

```
module mycomb ( cin1, cin2, cin3, cin4, cout1, cout2, cout3)
    /* synthesis syn_black_box = 1 */;
    input cin1, cin2, cin3, cin4;
    output cout1, cout2, cout3;
endmodule

module top( input combin1, combin2, combin3, combin4, output
q1, q2, q3);
```

```
mycomb U_comb (
    (* pin_att0=1 *).cin1(combin1),
    (* pin_att1="INPUT" *).cin2(combin2),
    (* pin_att2=1.2367 *).cin3(combin3),
    .cin4(combin4),
    (* pin_att3=32 *).cout1(q1),
    (* pin_att4=1'b0*).cout2(q2),
    .cout3(q3)
);
endmodule
```

For the RTL code above, the attributes forward-annotated to the output netlist (VM file) are shown below:

```
mycomb U_comb (
    (* pin_att0=1 *).cin1(combin1),
    (* pin_att1="INPUT" *).cin2(combin2),
    (* pin_att2=1.236700 *).cin3(combin3),
    .cin4(combin4),
    (* pin_att3=32 *).cout1(q1),
    (* pin_att4=0 *).cout2(q2),
    .cout3(q3)
);
```

Integration with Achronix Tools and Flows

After synthesis, the software generates a log file and output files for Achronix. For example, see:

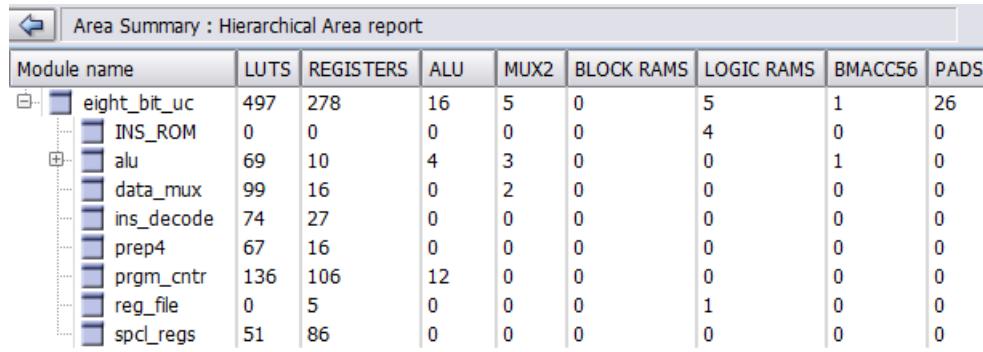
- [Hierarchical Area Report](#), on page 540
- [Running Post-Synthesis Simulation](#), on page 540
- [Designing for Achronix Architectures](#), on page 541

Hierarchical Area Report

The software generates a Hierarchical Area Report in the Project Status view of the Area Summary section. This support also provides a `csv` file located at

`/synlog/report/designName_fpga_mapper_hier_area.csv`

You can use this file to write your own scripts to generate reports.



Module name	LUTS	REGISTERS	ALU	MUX2	BLOCK RAMS	LOGIC RAMS	BMACC56	PADS
eight_bit_uc	497	278	16	5	0	5	1	26
INS_ROM	0	0	0	0	0	4	0	0
alu	69	10	4	3	0	0	1	0
data_mux	99	16	0	2	0	0	0	0
ins_decode	74	27	0	0	0	0	0	0
prep4	67	16	0	0	0	0	0	0
prgm_cntr	136	106	12	0	0	0	0	0
reg_file	0	5	0	0	0	1	0	0
spcl_regs	51	86	0	0	0	0	0	0

Running Post-Synthesis Simulation

For post-synthesis simulation with an Achronix design, do the following:

1. Run synthesis as usual.

The run generates a `.vhm` file, which references the `synplify` library:

```
library synplify;
use synplify.components.all;
```

2. Set up the libraries.
 - Create a library called synplify and compile synplify.vhd into it. The synplify.vhd file is located in *installDir/lib/vhdl_sim*.
 - Create a library called SPEEDSTER, and compile the Speedster simulation library provided by Achronix into it.
3. Compile the `vhm` file into work. For example:

```
vcom -work synplify installDir/lib/vhdl_sim/synplify.vhd
```

OR

1. Run synthesis as usual.
The run generates `a.vm` file.
2. Compile the `vm` file into work along with the Speedster simulation library provided by Achronix.

Designing for Achronix Architectures

The tips listed here are in addition to the technology-independent design tips described in [General Optimization Tips, on page 554](#).

- To ensure that frequency constraints from register to output pads are forward annotated to the P&R tools, add default `input_delay` and `output_delay` constraints of 0.0 in the synthesis tool. The synthesis tool forward-annotates the frequency constraints as PERIOD constraints (register-to-register) and OFFSET constraints (input-to-register and register-to-output). The place-and-route tools use these constraints.
- Run successive place-and-route iterations with progressively tighter timing constraints to get the best results possible.

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 the *Attribute Reference* manual; see [How Attributes and Directives are Specified](#), on page 10

Attribute/Directive	Description
<code>black_box_pad_pin</code>	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.
<code>black_box_tri_pins</code>	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.
<code>full_case</code>	Specifies that a Verilog case statement has covered all possible cases.
<code>loop_limit</code>	Specifies a loop iteration limit for for loops.
<code>parallel_case</code>	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
<code>pragma translate_off/pragma translate_on</code>	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
<code>syn_allow_retiming</code>	Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.
<code>syn_black_box</code>	Defines a black box for synthesis.
<code>syn_direct_enable</code>	Identifies which signal to use as the enable input to an enable flip-flop when multiple candidates are possible.
<code>syn_DSPstyle</code>	Determines whether the multiplier logical structures are mapped into MLP primitives or to logic.
<code>syn_encoding</code>	Specifies the encoding style for state machines.
<code>syn_enum_encoding</code>	Specifies the encoding style for enumerated types (VHDL only).

Attribute/Directive	Description
<code>syn_force_seq_prim</code>	Applies the “fix gated clocks” algorithm to the associated primitive.
<code>syn_gatedclk_clock_en</code>	Specifies the name of the enable pin inside a black box to support the “fix gated clocks” feature.
<code>syn_gatedclk_clock_en_polarity</code>	Indicates the polarity of the clock enable port on a black box, so that the software can apply the algorithm to fix gated clocks.
<code>syn_hier</code>	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
<code>syn_insert_pad</code>	Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.
<code>syn_isclock</code>	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
<code>syn_keep</code>	Prevents the internal signal from being removed during synthesis and optimization.
<code>syn_looplimit</code>	Specifies a loop iteration limit for while loops.
<code>syn_maxfan</code>	Overrides the default fanout guide for an individual input port, net, or register output.
<code>syn_max_memsize_reg</code>	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.
<code>syn_netlist_hierarchy</code>	Determines if the VMA output netlist is flat or hierarchical.
<code>syn_noarrayports</code>	Specifies signals as scalar in the output file.
<code>syn_noclockbuf</code>	Turns off automatic clock buffer usage.
<code>syn_no_compile_point</code>	Software automatically identifies modules as compile points in the design based on its size, number of I/Os, and hierarchical levels.
<code>syn_noprune</code>	Controls the automatic removal of instances that have outputs that are not driven.
<code>syn_pipeline</code>	Specifies that registers be moved into multipliers and ROMs in order to improve frequency.

Attribute/Directive	Description
<code>syn_preserve</code>	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
<code>syn_preserve_rom</code>	Apply on a ROM instance to control optimization when the ROM contents are read from the memory initialization file.
<code>syn_probe</code>	Adds probe points for testing and debugging.
<code>syn_ramstyle</code>	Determines the way in which RAMs are implemented.
<code>syn_reduce_controlset_size</code>	Controls the minimum size of the unique control-set on which control-set optimizations can occur.
<code>syn_reference_clock</code>	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
<code>syn_replicate</code>	Disables replication.
<code>syn_resources</code>	Specifies resources used in black boxes.
<code>syn_romstyle</code>	Determines how ROM architectures are implemented.
<code>syn_rw_conflict_logic</code>	Software automatically identifies modules as compile points in the design based on its size, number of I/Os, and hierarchical levels.
<code>syn_sharing</code>	Specifies resource sharing of operators.
<code>syn_shift_resetphase</code>	Allows you to remove the flip-flop on the inactive clock edge, built by the reset recovery logic for an FSM when a single event upset (SEU) fault occurs.
<code>syn_srlstyle</code>	Determines how to implement the sequential shift components.
<code>syn_state_machine</code>	Determines if the FSM Compiler extracts a structure as a state machine.
<code>syn_tco< n ></code>	Defines timing clock to output delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tpd< n ></code>	Specifies timing propagation for combinational delay through the black box. The <i>n</i> indicates a value between 1 and 10.

Attribute/Directive	Description
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 <i>n</i> indicates a value between 1 and 10.
syn_unconnected_inputs	Specifies that input pins of an instance be left floating (unassigned in the RTL).
syn_useenables	Prevents generation of registers with clock enable pins.
syn_useioff and ace_useioff	Supports forward annotation of the property syn_useioff=0 or 1 or ace_useioff = 0 or 1 in the netlist (.vm file).
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

APPENDIX B

Designing with Intel FPGA

The following topics describe how to design and synthesize with different Intel FPGA technologies:

- [Basic Support for Intel FPGA Designs](#), on page 548
- [Inputs for Intel FPGA Designs](#), on page 552
- [Intel FPGA Components](#), on page 554
- [Intel FPGA Constraints, Attributes, and Options](#), on page 599
- [Intel FPGA Device Mapping Options](#), on page 616
- [Integration with Intel FPGA Tools and Flows](#), on page 637
- [Intel FPGA Output Files and Forward Annotation](#), on page 649
- [Running Post-Synthesis Simulation with VHM Netlist](#), on page 656
- [Intel FPGA Place-and-Route Guidelines](#), on page 657
- [Intel FPGA Attribute and Directive Summary](#), on page 660

Basic Support for Intel FPGA Designs

This section describes general guidelines for using the synthesis tool and Intel FPGA place-and-route (P&R) tool with Intel FPGA devices. Refer to:

- [Intel FPGA Device Support](#), on page 548
- [Netlist Format](#), on page 549
- [Supported Synthesis Design Flows for Intel FPGA Designs](#), on page 550

Intel FPGA Device Support

The synthesis tools create technology-specific netlists for various Intel 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.

Family	Technologies
Arria	Arria10, Arria V GZ, Arria V, Arria II GX, Arria II GZ, Arria GX
Cyclone	Cyclone 10 GX/LP, Cyclone V, Cyclone IV GX, Cyclone IV E, Cyclone III LS, Cyclone III, Cyclone II, Cyclone
MAX 10	MAX 10
MAX	MAX 3000
MAX II	MAX V, MAX II
Stratix	Agilex, Stratix 10, Stratix V, Stratix IV, Stratix III, Stratix II GX, Stratix II, Stratix GX, Stratix HARDCOPY IV, HARDCOPY III, HARDCOPY II

See the following for device-specific information:

- [Stratix, Arria, and Cyclone Families](#), on page 621
- [MAX II, MAX V, MAX 10 Families](#), on page 624
- [MAX Family](#), on page 628

After synthesis, the tool generates netlists that are used as input to the Intel FPGA 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 place-and-route tool.

Netlist Format

The synthesis tool outputs VQM netlist files for use with the Intel FPGA P&R application. These files have a `.vqm` extension.

You can also use the project Tcl command to specify the result file format.

```
project -result_format vqm
```

To generate Intel FPGA output, see [Targeting Output for Intel FPGA, on page 549](#).

Targeting Output for Intel FPGA

You can generate output targeted for Intel FPGA.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

The following table summarizes the outputs to set for the different devices, and shows the P&R tools for which the output is intended.

Vendor Support	Output Netlist	P&R Tool
Intel FPGA (Arria 10)	Verilog (<code>.vqm</code>)	Quartus Prime (Pro Edition)
Intel FPGA (All devices except Arria 10)	Verilog (<code>.vqm</code>)	Quartus Prime (Standard)
Intel FPGA (All devices)	Verilog (<code>.vqm</code>)	Quartus II

3. To generate mapped Verilog netlists and constraint files, check the appropriate boxes and click OK.

Customizing Netlist Formats

The following table lists some attributes for customizing your Intel FPGA output netlists:

For ...	Use ...
Netlist formatting	<code>syn_netlist_hierarchy</code> <code>define_global_attribute syn_netlist_hierarchy {0}</code>
Bus specification	<code>syn_noarrayports</code> <code>define_global_attribute syn_noarrayports {1}</code>

Intel FPGA Forward Annotation

The synthesis tool generates Intel FPGA-compliant constraint files from selected constraints that are forward-annotated (read in and then used) by the Intel Quartus place-and-route software. The Intel FPGA constraint file uses the `.acf`, `.scf`, and `.tcl` extensions. This constraint file must be imported into the Intel FPGA flow.

By default, Intel FPGA 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).

Supported Synthesis Design Flows for Intel FPGA Designs

The following summarizes the synthesis design flows supported for Intel FPGA designs. All flows may not be available for all technologies:

Logic synthesis	See Logic Synthesis Design Flow, on page 38
Logic synthesis with floorplan	<i>Synplify Premier with Design Planner</i> See Design Plan-Based Logic Synthesis, on page 40 in the <i>User Guide</i> .
Logic synthesis with a synthesis strategy	<i>Synplify Premier</i> See Chapter 11, Setting Synthesis Strategies in the <i>User Guide</i> .

The synthesis tools also support other flows which allow better integration with the Intel FPGA components, flows, or back-end tools. See [Integration with Intel FPGA Tools and Flows, on page 637](#) for information.

Inputs for Intel FPGA Designs

The following describe how to handle different kinds of input for Intel FPGA designs:

- [Intel FPGA Verilog Libraries](#), on page 552
- [Intel FPGA VHDL Libraries](#), on page 552
- [IP Core Support](#), on page 552
- [QSF Constraints](#), on page 553

Intel FPGA Verilog Libraries

The synthesis tool automatically loads the Intel FPGA 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 Intel FPGA macro files are in *install_dir/lib/altera/quartusVersion*.

Intel FPGA VHDL Libraries

The Intel FPGA 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.

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 Intel FPGA IP:

Clearbox Megafunctions	The synthesis software can implement megafunctions generated by the Intel FPGA 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 688 of the <i>User Guide</i> .
SOPC Builder Cores	The synthesis tools can incorporate IP cores generated with SOPC Builder in Intel FPGA designs. For details of the procedure, see Including Intel FPGA Processor Cores from SOPC Builder, on page 707 of the <i>User Guide</i> .
Grey Box Megafunctions	You can incorporate encrypted IP as grey boxes in your Intel FPGA 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 Instantiating Megafunctions Using Grey Box Netlists, on page 699 in the <i>User Guide</i> .
SOPC Builder Components	The synthesis tool lets you use components generated with SOPC Builder in your Intel FPGA designs. For details of the procedure, see Working with SOPC Builder Components, on page 713 of the <i>User Guide</i> .

QSF Constraints

The synthesis tools include functionality to translate Intel FPGA QSF constraints to the SDC format. See [Translating Intel FPGA QSF Constraints, on page 267](#) in the *User Guide* for details.

Intel FPGA Components

The following topics describe how the synthesis tools handle various Intel FPGA components, and show you how to work with or manipulate them during synthesis to get the results you need:

- [Intel FPGA Macros](#), on page 554
- [Intel FPGA Black Boxes](#), on page 555
- [Intel FPGA ATOM Primitives](#), on page 555
- [Intel FPGA Multipliers](#), on page 556
- [Special Registers in Intel FPGA Designs](#), on page 556
- [Intel FPGA PLLs](#), on page 556
- [Stratix ShiftTaps](#), on page 556
- [Intel FPGA Pad Cells](#), on page 557
- [Intel FPGA RAM and ROM Implementations](#), on page 557
- [Intel FPGA DSP Blocks](#), on page 567
- [Intel FPGA LPMs](#), on page 593

Intel FPGA Macros

You can select macros from the Intel FPGA 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 Intel FPGA designs. Refer to the Intel FPGA documentation for supported macros.

Verilog	<p>For Verilog designs, the tool automatically adds the Verilog component declaration file for the target technology to your project from the <code>lib/altera/quartusVersion</code> 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.</p> <p>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 <code>stratixiii.v</code> file from the <code>quartus_II111</code> directory to access the port and parameter definitions for the primitives.</p>
VHDL	<p>For VHDL designs, instantiate Intel FPGA 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:</p> <pre>library <i>family</i> ; use altera.<i>family</i>.all;</pre> <p>In this example, <i>family</i> is the name of the library family such as <code>stratixiigx</code> or <code>cycloneii</code> (for a list of family names, see the <code>location.map</code> file in the <code>/lib/vhd</code> directory).</p>

Intel FPGA Black Boxes

You can create an instance of any externally defined macro, including a user-defined macro or Intel FPGA 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 514](#) in the *User Guide*.

Intel FPGA 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.

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 Intel FPGA device families, and more accurately estimate timing.

Intel FPGA 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 Intel FPGA Designs

The compiler can infer additional register primitive types so that the mapper can implement a better solution, when it targets Intel FPGA 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 <code>dffe</code> , <code>dffe</code> , <code>dfse</code> , <code>dfrse</code> , <code>dffpatre</code> and <code>dffpatre</code> . Set <code>syn_direct_enable</code> 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 189 . 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 <code>sdffr</code> , <code>sdfss</code> , <code>sdfrrs</code> , <code>sdfppat</code> , <code>sdfre</code> , <code>sdfse</code> and <code>sdfppate</code> . The compiler does this automatically and does not require a directive.

Intel FPGA PLLs

The synthesis software recognizes Intel FPGA PLL components, or `altplls`. You generate the component files with the Intel FPGA MegaWizard tool and then add constraints before you synthesize. For details of the procedure, see [Working with Intel FPGA PLLs, on page 640](#).

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 537](#) of the *User Guide* for information on implementing these components.

Intel FPGA 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 ...	<code>cellName, any</code>
To insert a pad cell on a specific pin, use ...	<code>cellName, pinName</code>
To insert pad cells on all inputs, use ...	<code>cellName, input</code>
To insert pad cells on all outputs, use ...	<code>cellName, output</code>

For example:

```
altsyncram, any  
test0, in0  
test1, input  
test2, output\
```

Intel FPGA RAM and ROM Implementations

The following describe how the synthesis tools infer and implement Intel FPGA RAM and ROM. For information about other Intel FPGA components, see [Intel FPGA Components, on page 554](#).

- [Intel FPGA RAM, on page 558](#)

- [Stratix, Arria, and Cyclone RAMs](#), on page 558
- [Asymmetric RAM](#), on page 560
- [Byte-Wide Write Enable RAM](#), on page 560
- [Intel FPGA RAM with Control Signals](#), on page 564
- [Initial Values for Intel FPGA RAMs](#), on page 565
- [Intel FPGA ROM](#), on page 565
- [Intel FPGA MIF Files](#), on page 567

See [Intel FPGA LPMs](#), on page 593, and [Intel FPGA DSP Blocks](#), on page 567 for descriptions of other Intel FPGA components.

Intel FPGA RAM

From the HDL source code, the synthesis tool automatically infers RAMs and generates single-port or dual-port RAMs. This section contains an overview; for details refer to the RAM inference sections in [Chapter 6, RAM and ROM Inference](#).

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).

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 <code>syn_ramstyle</code> value to <code>no_rw_check</code> .
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 509](#)) 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
Agilex, Stratix 10, Stratix V, Cyclone V, Arria V, Arria 10, Cyclone 10GX	M20K
Cyclone, Cyclone II	M4K
Cyclone III, Cyclone IV E, Cyclone IV GX, Cyclone 10 LP	M9K
Arria GX, Arria II GX	MRAM, M512

For information on ROM inference for Stratix, Arria, and Cyclone, see [Intel FPGA ROM, on page 565](#).

Intel FPGA RAM DRC Restrictions

Arria 10, Arria V, Cyclone V, Stratix V, Stratix 10, and Agilex families

When you configure both ports of a true dual-port RAM with same port read during write in OLD_DATA (READ_FIRST) mode, the Intel Quartus II tool returns a DRC restriction for the RAM. The synthesis tool does not support this true dual-port RAM configuration and cannot pack the RAM into a single RAM block (for example, M20K).

As a workaround, modify the HDL code for the true dual-port RAM by configuring one of the ports with same port read during write in NEW_DATA_NO_NBE_READ (WRITE_FIRST) mode. Then, the synthesis tool can map the RAM into one RAM block similarly to place and route.

Asymmetric RAM

Arria V and Stratix V families

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.

For details, see [Asymmetric RAM Inference, on page 269](#) and [Asymmetric RAM Examples, on page 269](#) in the *Reference Manual*.

Byte-Wide Write Enable RAM

Stratix V family

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 Intel stratixv_ram_block or MLABs for improved performance by merging the 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 509](#) for more information.

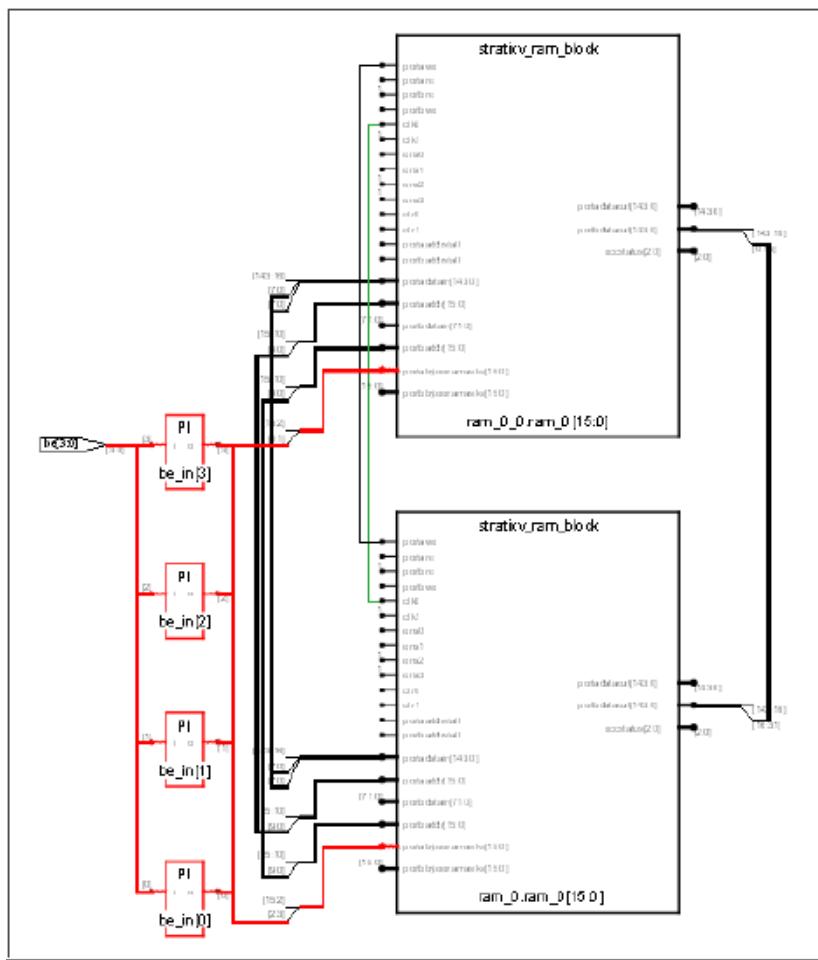
For examples, see:

- [Simple Dual-port RAM Example](#)

- [Simple Dual-port RAM \(SystemVerilog Compatible\) Example](#)
- [Single-port Write First RAM Example](#)

Simple Dual-port RAM Example

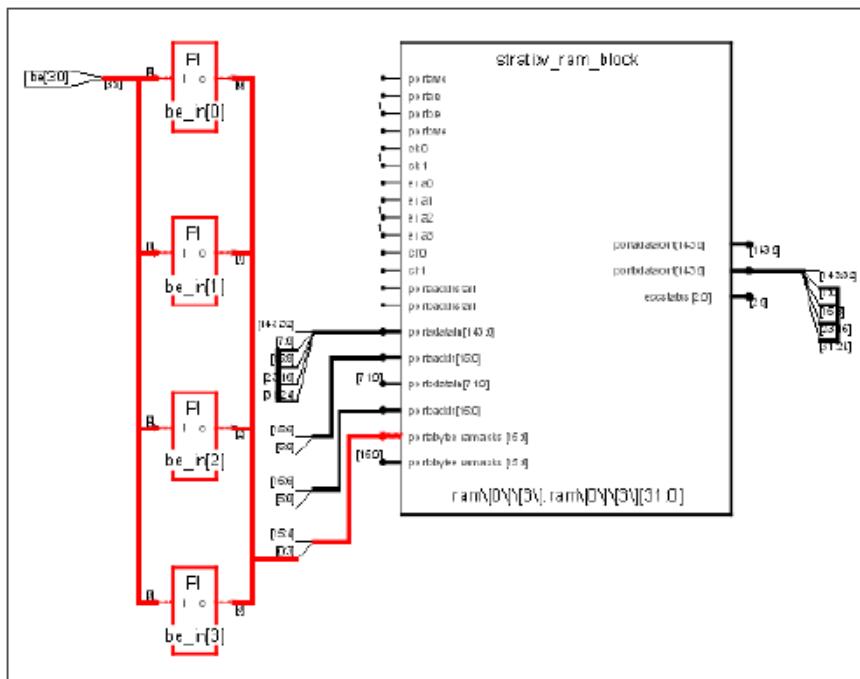
In this example, the RTL byte-wide write enable RAM are shown mapped to the Intel `stratixv_ram_block` component in the Technology view. The two `stratixv_ram_block` blocks are generated. These blocks were filtered and the `portabtreenamask` for the blocks were expanded to register/ports. This view shows 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.



Simple Dual-port RAM (SystemVerilog Compatible) Example

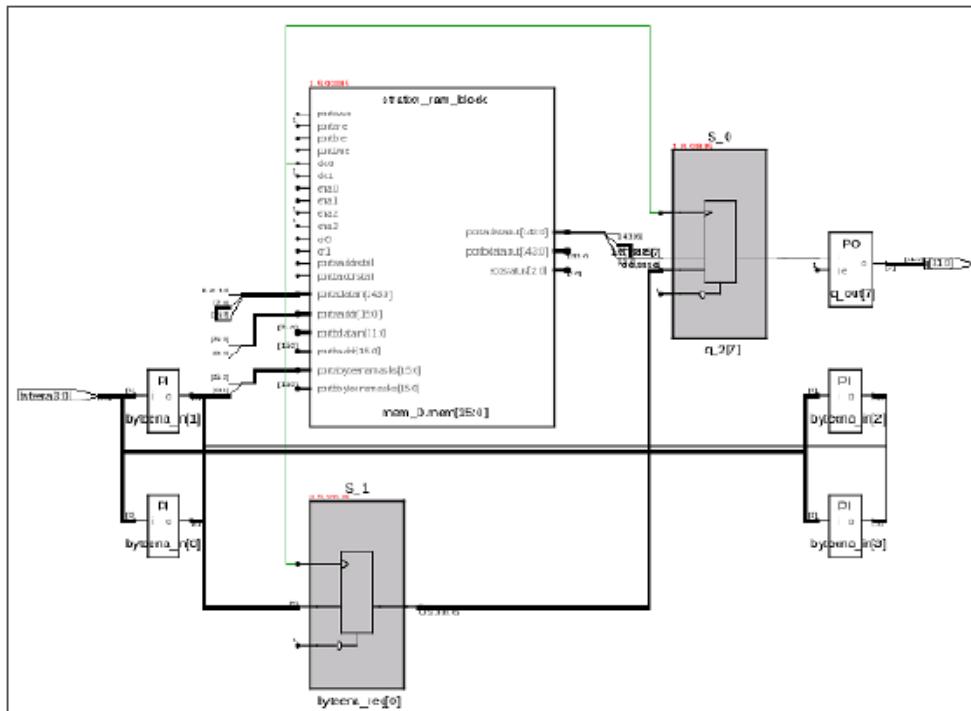
This SystemVerilog example is equivalent to Example 1 above, except that it contains a 3-dimensional RAM with byte-wide write enable. For 3-dimensional RAM, make sure to enable the System Verilog switch on the Verilog tab of the Implementation Options panel.

In this example, the RTL byte-wide write enable RAM are shown mapped to the Intel stratixv_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.



Single-port Write First RAM Example

In the Technology view below, this portion of the design shows an example of Intel 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.



Intel FPGA RAM with Control Signals

Stratix III and Stratix IV families

The enable pins on Intel FPGA 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.

Cyclone III family

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.

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 509](#).

See [RAM Inference with Control Signals, on page 257](#) in the *User Guide* and [RAM with Control Signals Examples, on page 258](#) for details and examples.

Initial Values for Intel FPGA RAMs

You can specify startup values for RAMs in Intel FPGA designs. See [Initializing RAMs, on page 528](#) in the *User Guide* for step-by-step procedures on setting initial values in Verilog or VHDL code.

Initial Value Limitations for Intel FPGA RAM

Note the following RAM2 initialization limitation.

Intel FPGA RAM2 Initialization Issue Can Cause Simulation Mismatches

Initialized values for an Intel FPGA core resettable RAM (RAM2) with asynchronous reset cannot be read out and are ignored. This can cause simulation mismatches, as mentioned in the warning message that is generated:

```
RAM mem[7:0] has asynchronous core reset along with non-zero
initial value that Intel FPGA devices do not support. Initial value
is ignored. Simulation mismatch is possible.
```

As a workaround, design the RAM using a synchronous core reset.

Intel FPGA ROM

ROMs that would normally be implemented as logic can be implemented as RAMs. See [syn_romstyle, on page 577](#) for syntax details.

The following table shows how ROMs are inferred and implemented in different Intel FPGA technologies.

Technology	Synthesis infers ...	Implementation
Stratix	<i>technology_ram_block</i>	RAM. A synchronous ROM can have any of the following configurations:
Stratix GX	RAM primitive (for	<ul style="list-style-type: none"> • Only read address registered
Stratix II	synchronous ROMs	<ul style="list-style-type: none"> • Only ROM output registered
HardCopy II	only).	<ul style="list-style-type: none"> • Both read address and ROM output registered
Stratix II GX		
Stratix III		
HardCopy III		
Stratix IV		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.
HardCopy IV		
Stratix V		
Stratix 10		
Agilex		
Cyclone		
Cyclone II		
Cyclone III		
Cyclone IV E		
Cyclone IV GX		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.
Cyclone V		
Arria GX		
Arria II GX		
Arria II GZ		
Arria V		
Arria 10		
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.

Intel FPGA MIF Files

The Intel FPGA MegaWizard generates a RAM or ROM, which has an instantiated ALTSYNCRAM and defparam init_file that specifies a memory initialization file (MIF). The synthesis tools search for these MIF files in their referenced directories, and then copy them to the implementation and integrated place-and-route directories. The Intel FPGA place-and-route software uses the VQM file with these MIF files.

When synthesizing the RAM or ROM, if a MIF is missing there are two ways to handle this problem:

1. Supply the path to the missing MIF files. You can:
 - Select the MIF File Directories option on the Device tab of the Implementation Options panel.
 - Specify the absolute directory location of the MIF file in the Value field (for example, "U:/mif_tests"). Separate multiple directory entries with a semicolon.
2. Do not check for MIF files when synthesizing the RAM or ROM. You can:
 - Disable the Validate MIF Files setting. To do this:
Disable (uncheck) the Validate MIF Files option from the Options tab of the Implementation Options panel or use the Tcl project command:
`set_option -validate_mif_files 0`
 - Provide the MIF files for Quartus place and route.

Intel FPGA DSP Blocks

Intel Stratix, Arria, and Cyclone Families

These 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 Intel FPGA 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 568

- [Attributes and Directives for DSP Block Inference](#), on page 591
- [Synchronous and Asynchronous Reset Packing for DSP Blocks](#), on page 592

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: <ul style="list-style-type: none">• 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

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);
```

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
```

The `altsmult_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 `altsmult_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.

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
```

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 `altsync_mmult` megafunction when possible.

It implements the `altsync_mmult` 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 `altsync_mmult` megafunction for scan chains formed by the input registers of an accumulator.

Multiply-accumulate mode with synchronous load

The `altsync_mmult` 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.

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 `almtult_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.

Pre-adder/subtractor

Pre-adder/subtractor driving a multiplier

```
x = (a + b)*c + (d + e)*f;
x = (a + b)*c;
x = (a - b)*c + (d - e)*f;
x = (a - b)*c;
```

Stratix V, Arria V GZ

The multiplier uses two pre-adder 25-bit inputs that supports `27x27` and `27x27_sumof2` multiplications.

The dynamic input of the multiplier must be restricted to 22 bits.

Arria V, Cyclone V

The multiplier uses two pre-adder 17-bit (signed/unsigned) or 18-bit (signed) inputs for `18x18`, `18x18_sumof2`, `18x18_systolic` multiplications and 26-bit inputs for `27x27` multiplications.

The tool packs pre-adder/subtractor into the DSP.

Configuration

Internal coefficient

Internal coefficient storage driving a multiplier that supports up to eight constant coefficients of 18-bits for 18x18 multiplication or 27-bits for 27x27 multiplication.

```
x = (a + b)*coeff;
x = (a + b)*coeff1 + (c + d)*coeff2;
x = (a - b)*coeff;
x = (a - b)*coeff1 + (c - d)*coeff2;
x = a*coeff;
x = a*coeff1 + b*coeff2;
```

The synthesis tool implements ...

Stratix V, Arria V GZ

The multiplier uses internal coefficient storage for m18x18_full, m18x18_sumof2, m18x18_systolic, m18x18_sumof4, m27x27, and m27x27_sumof2 multiplications.

For a multiplier with a pre-adder, the input to the pre-adder is restricted to 26 bits.

Arria V, Cyclone V

The multiplier uses internal coefficient storage for m18x18_full, m18x18_sumof2, m18x18_systolic, and m27x27 multiplications.

For a multiplier with a pre-adder, the input to the pre-adder is restricted to 26 bits.

The tool packs coefficient ROM into the DSP.

Multiplier-add accumulate

Multiplier driving an accumulator

```
x = x + (a*b);
```

Multiplier adder driving an accumulator

```
x = x + (a*b) + (c*d);
x = x + (a*b) + c;
```

Stratix V, Arria V GZ

The multiplier uses a 64-bit accumulator for m27x27, m36x18, m18x18_sumof2, m18x18_plus36 multiplications and 44-bit accumulator for m18x18_systolic multiplications.

Arria V, Cyclone V

The multiplier uses a 64-bit accumulator for m27x27, m18x18_sumof2, m18x18_plus36 multiplications and 44-bit accumulator for m18x18_systolic multiplications.

Double accumulation provides an extra register in the feedback path of the accumulator for the multiplier and multiplier-add accumulate.

The tool packs the multiplier-add accumulator into a DSP for the scenarios above.

Configuration**The synthesis tool implements ...****Dynamic Adder/Subtractor**

The "sub" control signal for an Arria V, Cyclone V, and Stratix V DSP block can be used to control add/subtract operations dynamically. The "sub" pin controls addition or subtraction of two 18x18 multiplier results.

Stratix V, Arria V GZ

Uses a 36-bit adder/subtractor for m36x18, m18x18_sumof2, m18x18_plus36, m18x18_systolic multiplications and 54-bit adder/subtractor for m27x27 and m27x27_sumof2 multiplications.

Arria V, Cyclone V

Uses a 36-bit adder/subtractor for m18x18_sumof2, m18x18_plus36, m18x18_systolic multiplications and 54-bit adder/subtractor for m27x27 multiplication.

The tool packs dynamic adder/subtractor logic into the DSP for the scenarios above.

Scan Chain Inference

Scan chain registers driving multipliers can be inferred and packed in the DSP.

Stratix V

Supports the following DSP configurations: m18x18_sumof2, m18x18_systolic, m18x18_sumof4, m27x27, m27x27_sumof2 multiplications.

Arria V, Cyclone V

m18x18_sumof2, m18x18_systolic, m27x27 multiplications.

The tool identifies scan chain registers and packs them into the DSP.

The following examples show how synthesis automatically infers Intel FPGA DSP blocks.

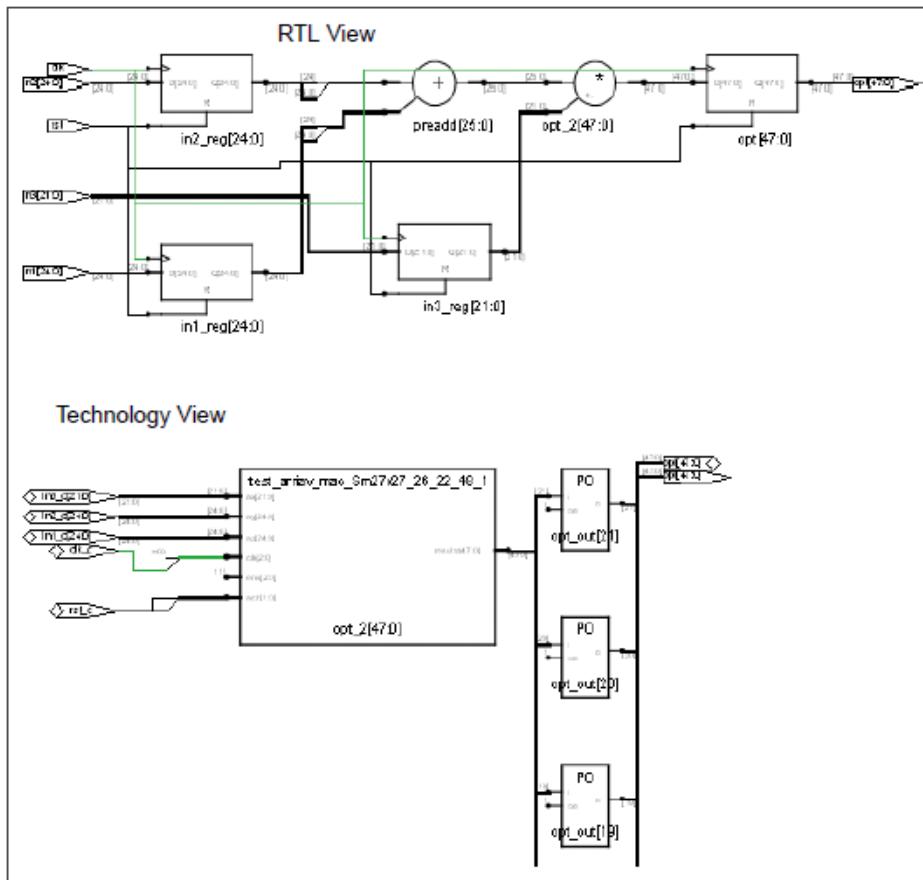
Example 1: Pre-adder/Subtractor for M27x27 Multiplication

```
module test (clk, rst, in1, in2, in3, opt);
    input clk;
    input rst;
    input signed [24: 0] in1, in2
```

```
input signed [21: 0] in3;
output reg signed [47 : 0] opt;

reg signed [24 : 0] in1_reg;
reg signed [24 : 0] in2_reg;
reg signed [21 : 0] in3_reg;
wire signed [25 : 0] preadd;
assign preadd = (in1_reg + in2_reg); //preadd = (in1_reg -
in2_reg); for subtraction
always @ (posedge clk or posedge rst)
begin
    if(rst)
        begin
            in1_reg <= 0;
            in2_reg <= 0;
            in3_reg <= 0;
            opt <= 0;
        end
    else
        begin
            in1_reg <= in1;
            in2_reg <= in2;
            in3_reg <= in3;
            opt <= preadd * in3_reg;
        end
    end
endmodule
```

See the following RTL and Technology views.



Example 2: Internal Coefficient Storage for Two M18x18 Multiplication with Systolic Delay Registers

```
module test (clk, rst, en, in1, in2, in3, in4, ca_sel, cb_sel,
opt);

input clk,rst,en;
input [2:0] ca_sel,cb_sel; // Coeffcient storage select signals
input signed [17 : 0] in1, in3;
input signed [17 : 0] in2, in4;
```

```
output reg signed [37 : 0] opt;

reg signed [17 : 0] in1_reg, in3_reg;
reg signed [17 : 0] in2_reg, in4_reg;

reg [2:0] ca_sel_reg,cb_sel_reg;

wire signed [18 : 0]preadd1;
wire signed [18 : 0]preadd2;
wire signed [36:0] mult1;
reg signed [36 :0] mult1_temp;
wire signed [36 :0] mult2;
reg signed [17 : 0] c1,c2;

always @(ca_sel_reg)
begin
  case (ca_sel_reg)
    3'b000 : c1=18'b100101001001111101;
    3'b001 : c1=18'b01111111110110110;
    3'b010 : c1=18'b101010101111010111;
    3'b011 : c1=18'b011101111010101100;
    3'b100 : c1=18'b010101111010101110;
    3'b101 : c1=18'b001101111101101101;
    3'b110 : c1=18'b010001111101101101;
    3'b111 : c1=18'b000101111100101110;
  endcase
end
```

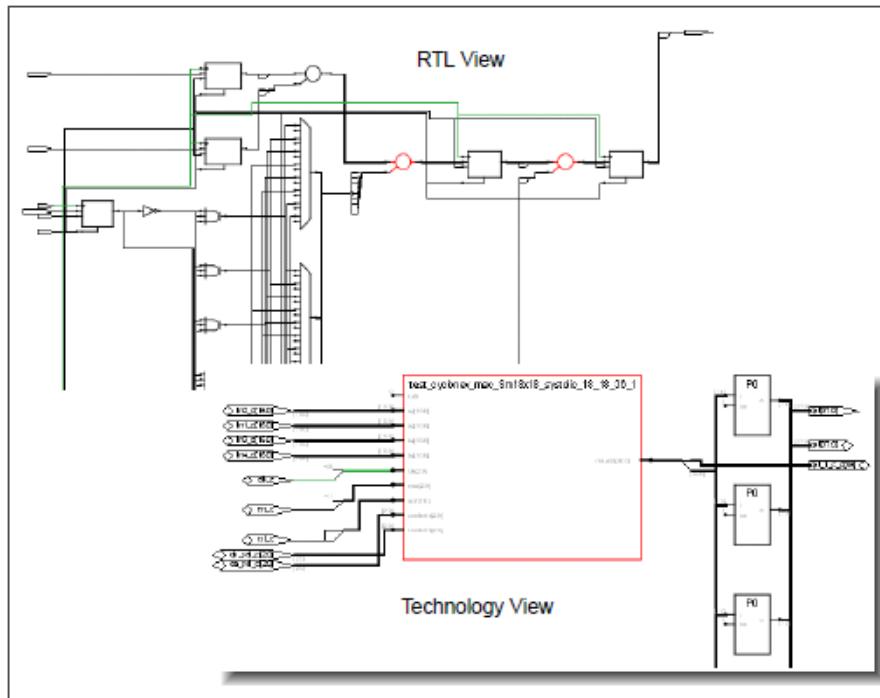
```
always @ (cb_sel_reg)
begin
  case (cb_sel_reg)
    3'b000 : c2=18'b110010100001111011;
    3'b001 : c2=18'b10111111110110101;
    3'b010 : c2=18'b110101010111010111;
    3'b011 : c2=18'b101110111010101001;
    3'b100 : c2=18'b101010111010101101;
    3'b101 : c2=18'b100110111101101011;
    3'b110 : c2=18'b101000111101101011;
    3'b111 : c2=18'b100010111100101101;
  endcase
end

assign preadd1 = (in1_reg + in2_reg);
assign preadd2 = (in3_reg + in4_reg);
assign mult1 = (preadd1 * c1);
assign mult2 = (preadd2 * c2);

always @ (posedge clk or posedge rst)
begin
  if (rst)
    begin
      in1_reg <= 0;
      in2_reg <= 0;
      in3_reg <= 0;
```

```
    in4_reg <= 0;  
    ca_sel_reg <= 0;  
    cb_sel_reg <= 0;  
    mult1_temp <= 0;  
    opt <= 0;  
  
end  
  
else  
  
if (en)  
  
begin  
  
    in1_reg <= in1;  
    in2_reg <= in2;  
    in3_reg <= in3;  
    in4_reg <= in4;  
    ca_sel_reg <= ca_sel;  
    cb_sel_reg <= cb_sel;  
    mult1_temp <= mult1;  
    opt <=mult1_temp+ mult2;  
  
end  
  
end  
  
endmodule
```

See the following RTL and Technology views.



Example 3: Accumulator for M18x18 Multiplication

```
module test (clk, rst, in1, in2, in3, in4, opt);
parameter mult_ip1=18;
parameter mult_ip2=18;
parameter op_width= 63;
// enable_double_accum_reg = 1 to infer Double accumulation
register for ArriaV/CycloneV
parameter enable_double_accum_reg = 0;
input clk,rst;
input [mult_ip1- 1 : 0] in1, in3;
```

```
input [mult_ip2- 1 : 0] in2, in4;
output reg [op_width-1 : 0] opt;
reg [op_width-1 : 0] double_accum_reg;

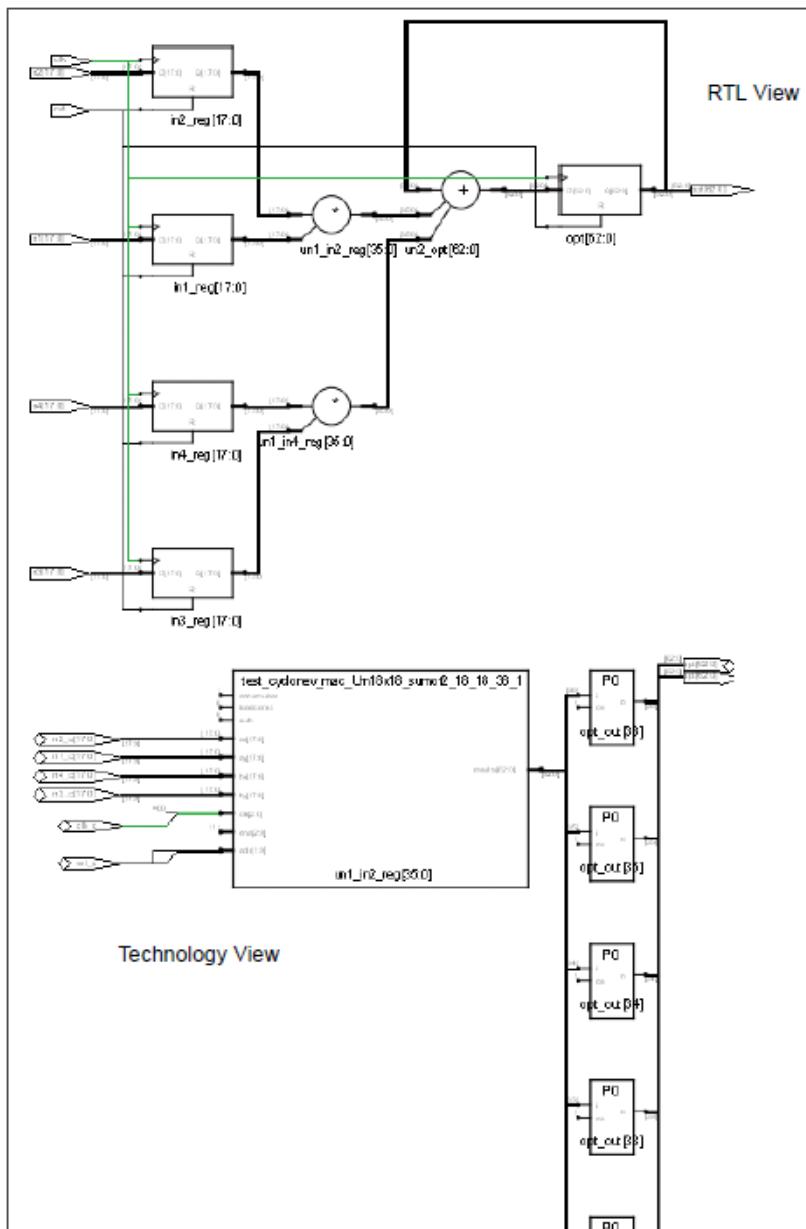
reg [mult_ip1- 1 : 0] in1_reg, in3_reg;
reg [mult_ip2- 1 : 0] in2_reg, in4_reg;

wire [mult_ip1+mult_ip2 :0] mult_add;
assign mult_add = in1_reg * in2_reg + in3_reg * in4_reg;

always @ (posedge clk or posedge rst)
begin
    if (rst)
        begin
            in1_reg <= 0;
            in2_reg <= 0;
            in3_reg <= 0;
            in4_reg <= 0;
            double_accum_reg <= 0;
            opt <= 0;
        end
    else
        begin
            in1_reg <= in1;
            in2_reg <= in2;
            in3_reg <= in3;
            in4_reg <= in4;
        end
end
```

```
    opt <=accm_cntrl+ mult_add;
    double_accum_reg <= opt;
end
end
wire [op_width-1:0] accm_cntrl;
assign accm_cntrl = (enable_double_accum_reg == 1) ?
double_accum_reg : opt;
endmodule
```

See the following RTL and Technology views.



Example 4: Dynamic Adder/Subtractor for M18x18 Sum of Two Multiplications

```
module test (clk,rst,en,in1,in2,in3,in4,addnsub,dout);  
parameter mult_ip_18= 18;  
parameter mult_op_36= 2*mult_ip_18;  
parameter add_sub_op_37 = mult_op_36+1;  
parameter op_width = add_sub_op_37;  
  
input clk;  
input rst,en;  
input addnsub; // Dynamic add-sub input  
  
input signed [mult_ip_18-1 : 0] in1;  
input signed [mult_ip_18-1 : 0] in2;  
  
input signed [mult_ip_18-1 : 0] in3;  
input signed [mult_ip_18-1 : 0] in4;  
  
output signed [op_width-1 : 0] dout;  
  
reg signed [mult_ip_18-1 : 0] in1_reg;  
reg signed [mult_ip_18-1 : 0] in2_reg;  
  
reg signed [mult_ip_18-1 : 0] in3_reg;  
reg signed [mult_ip_18-1 : 0] in4_reg;  
  
reg signed [op_width-1 : 0] dout;
```

```
        wire signed [mult_op_36-1 : 0] m1;
        wire signed [mult_op_36-1 : 0] m2;
        wire signed [add_sub_op_37-1  : 0] add;

    always @ (posedge clk or posedge rst)
begin
    if(rst)
begin
    in1_reg <= 0;
    in2_reg <= 0;
    in3_reg <= 0;
    in4_reg <= 0;
end
else if(en)
begin
    in1_reg <= in1;
    in2_reg <= in2;
    in3_reg <= in3;
    in4_reg <= in4;
end
end
assign m1 = in1_reg * in2_reg;
assign m2 = in3_reg * in4_reg;
// dynamic add/sub logic
assign add = addnsub ? (m1+m2) :( m1-m2);
always @ (posedge clk or posedge rst)
```

```
begin
    if(rst)
        begin
            dout <= 0;
        end
    else if (en)
        begin
            dout <= (add);
        end
    end

endmodule
```

See the following RTL and Technology views.

Example 5: Scan Chain Inference (Stratix V)

```
`ifdef synthesis
module test(clk,rst1, rst2, a_ip, b_ip, c_ip, d_ip, e_ip, dout);
`else
module test_rtl(clk,rst1, rst2, a_ip, b_ip, c_ip, d_ip, e_ip,
dout);
`endif
parameter IN_WIDTH=18;
parameter OUT_WIDTH=37;

input clk,rst1, rst2;
input [IN_WIDTH - 1 : 0] a_ip;
input [IN_WIDTH - 1 : 0] b_ip ;
input [IN_WIDTH - 1 : 0] c_ip ;
```

```
input [IN_WIDTH - 1 : 0] d_ip ;
input [IN_WIDTH - 1 : 0] e_ip ;
output reg [OUT_WIDTH:0] dout;

reg [IN_WIDTH - 1 : 0] b_ip_ff ;
reg [IN_WIDTH - 1 : 0] c_ip_ff ;
reg [IN_WIDTH - 1 : 0] d_ip_ff ;
reg [IN_WIDTH - 1 : 0] e_ip_ff ;

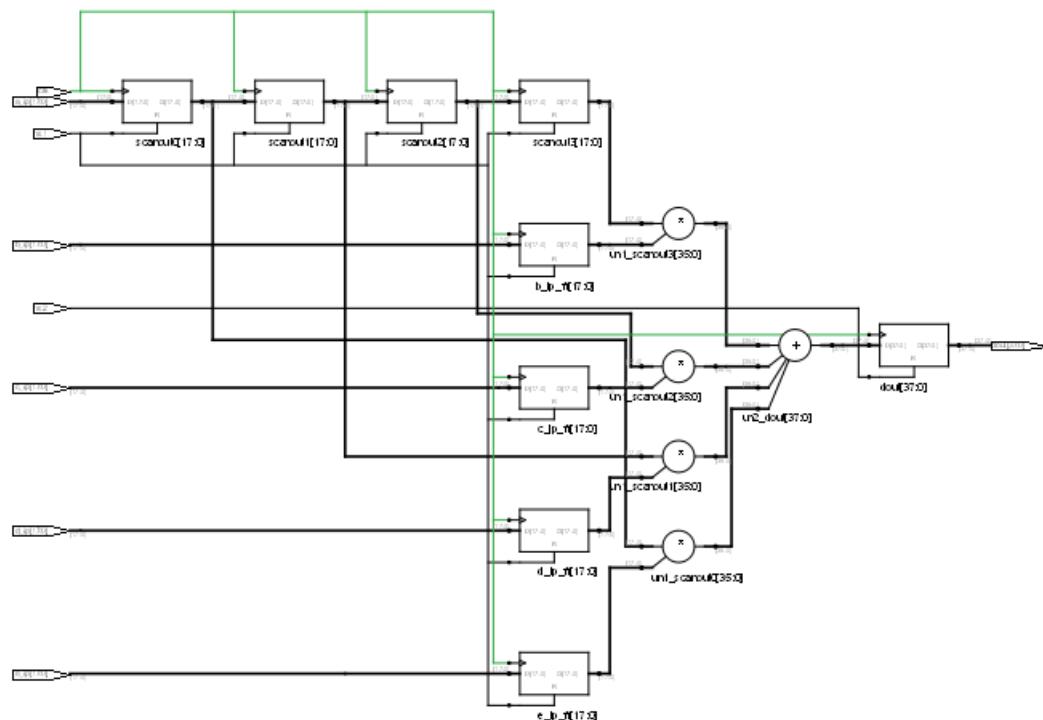
reg [IN_WIDTH-1:0] scanout0, scanout1, scanout2, scanout3;

always @ (posedge clk or posedge rst1)
begin
if (rst1) begin
    scanout0 <= {IN_WIDTH{1'b0}};
    scanout1 <= {IN_WIDTH{1'b0}};
    scanout2 <= {IN_WIDTH{1'b0}};
    scanout3 <= {IN_WIDTH{1'b0}};
    b_ip_ff <= {IN_WIDTH{1'b0}};
    c_ip_ff <= {IN_WIDTH{1'b0}};
    d_ip_ff <= {IN_WIDTH{1'b0}};
    e_ip_ff <= {IN_WIDTH{1'b0}};
end
else
begin
    scanout0 <= a_ip;
    scanout1 <= scanout0;
```

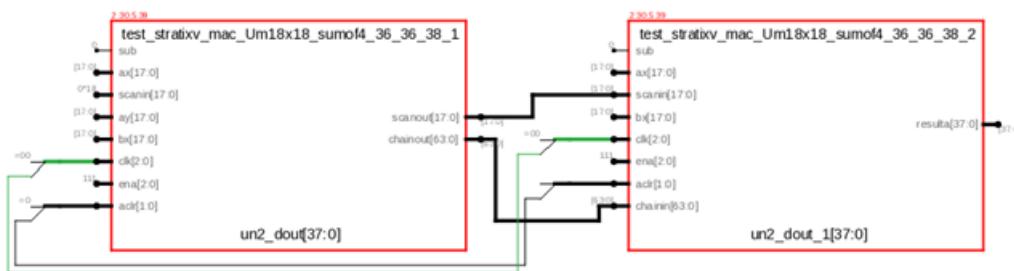
```
    scanout2 <= scanout1;  
    scanout3 <= scanout2;  
    b_ip_ff <= b_ip;  
    c_ip_ff <= c_ip;  
    d_ip_ff <= d_ip;  
    e_ip_ff <= e_ip;  
  
end  
end  
  
always @ (posedge clk or posedge rst2)  
begin  
    if (rst2)  
        dout <= {OUT_WIDTH{1'b0}};  
    else  
        dout <= (b_ip_ff*scanout3) + (c_ip_ff*scanout2) +  
        (d_ip_ff*scanout1) + (e_ip_ff*scanout0) ;  
end  
endmodule
```

This example shows how scan chain registers can be inferred and packed into the M18x18 SUMOF4 DSP with a Stratix V device.

The following RTL view shows the scan chain registers that drive multipliers.



The tool packs the scan chain registers into the m18x18_sumof4 DSP configuration blocks shown in the Technology view below.



Example 6: Scan Chain Inference (Arria V)

```
`ifdef synthesis
```

```
module test(clk,rst1, rst2, a_ip, b_ip, c_ip, d_ip, e_ip, dout);  
`else  
module test_rtl(clk,rst1, rst2, a_ip, b_ip, c_ip, d_ip, e_ip,  
dout);  
`endif  
parameter IN_WIDTH=18;  
parameter OUT_WIDTH=37;  
  
input clk,rst1, rst2;  
input [IN_WIDTH - 1 : 0] a_ip;  
input [IN_WIDTH - 1 : 0] b_ip ;  
input [IN_WIDTH - 1 : 0] c_ip ;  
input [IN_WIDTH - 1 : 0] d_ip ;  
input [IN_WIDTH - 1 : 0] e_ip ;  
output reg [OUT_WIDTH:0] dout;  
  
reg [IN_WIDTH - 1 : 0] b_ip_ff ;  
reg [IN_WIDTH - 1 : 0] c_ip_ff ;  
reg [IN_WIDTH - 1 : 0] d_ip_ff ;  
reg [IN_WIDTH - 1 : 0] e_ip_ff ;  
  
reg [IN_WIDTH-1:0] scanout0, scanout1, scanout2, scanout3;  
  
always @ (posedge clk or posedge rst1)  
begin  
if (rst1) begin  
    scanout0 <= {IN_WIDTH{1'b0}};
```

```
scanout1 <= {IN_WIDTH{1'b0}};  
scanout2 <= {IN_WIDTH{1'b0}};  
scanout3 <= {IN_WIDTH{1'b0}};  
b_ip_ff <= {IN_WIDTH{1'b0}};  
c_ip_ff <= {IN_WIDTH{1'b0}};  
d_ip_ff <= {IN_WIDTH{1'b0}};  
e_ip_ff <= {IN_WIDTH{1'b0}};  
  
end  
else  
begin  
    scanout0 <= a_ip;  
    scanout1 <= scanout0;  
    scanout2 <= scanout1;  
    scanout3 <= scanout2;  
    b_ip_ff <= b_ip;  
    c_ip_ff <= c_ip;  
    d_ip_ff <= d_ip;  
    e_ip_ff <= e_ip;  
  
end  
end  
  
always @ (posedge clk or posedge rst2)  
begin  
    if (rst2)  
        dout <= {OUT_WIDTH{1'b0}};
```

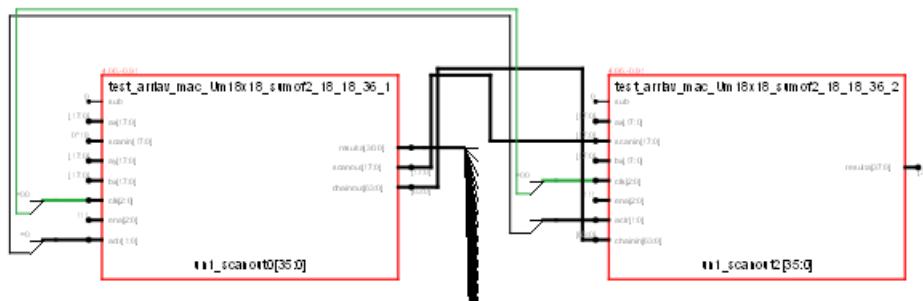
```

    else
        dout <= (b_ip_ff*scanout3) + (c_ip_ff*scanout2) +
        (d_ip_ff*scanout1) + (e_ip_ff*scanout0) ;
    end
endmodule

```

This example shows how scan chain registers can be inferred and packed into the M18x18 SUMOF2 DSP with an Arria V device.

The RTL view is the same as Example 5 above, which shows the scan chain registers driving multipliers. The tool packs the scan chain registers into the m18x18_sumof2 DSP configuration blocks shown in the Technology view below.



Attributes and Directives for DSP Block Inference

DSP block inference is influenced by the following attributes and directives:

syn_useoff	Registers with either of these attributes are not packed into MACs.
syn_preserve	
syn_keep	A syn_keep directive between the adder and the multiplier prevents inference of multipliers almtmult_add and almtmult_accum , but not plain multipliers.
syn_multstyle	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.

Synchronous and Asynchronous Reset Packing for DSP Blocks

The software packs asynchronous and synchronous resets into the Intel FPGA 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;

```

Intel FPGA LPMs

The Intel FPGA 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 543](#) in the *User Guide*.

There are three methods for instantiating Intel FPGA LPMs:

- As black boxes (Verilog or VHDL). Start by defining the LPM using the Intel FPGA 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 Intel FPGA LPMs as Black Boxes, on page 544](#) in the *User Guide* for details.

- Using the Verilog LPM library provided. See [Instantiating Intel FPGA LPMs Using a Verilog Library, on page 550](#) 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 594](#) 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 Intel FPGA place-and-route documentation for ports and generics that require mapping. See [Instantiating Intel FPGA LPMs Using VHDL Prepared Components, on page 548](#) 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.

<code>altcam_syn</code>	<code>altcdr_rx</code>	<code>altcdr_tx</code>	<code>altclklock</code>
<code>altclkclock_syn</code>	<code>altddio_bidir</code>	<code>altddio_in</code>	<code>altddio_out</code>
<code>altdpram</code>	<code>altlvds_rx</code>	<code>altlvds_tx</code>	<code>altqpram</code>
<code>csfifo</code>	<code>dcfifo</code>	<code>lpm_abs</code>	<code>lpm_add_sub</code>
<code>lpm_and</code>	<code>lpm_bustri</code>	<code>lpm_clshift</code>	<code>lpm_compare</code>
<code>lpm_components</code>	<code>lpm_constant</code>	<code>lpm_counter</code>	<code>lpm_decode</code>
<code>lpm_divide</code>	<code>lpm_ff</code>	<code>lpm_fifo</code>	<code>lpm_fifo_dc</code>
<code>lpm_inv</code>	<code>lpm_latch</code>	<code>lpm_mult</code>	<code>lpm_mux</code>
<code>lpm_ram_dp</code>	<code>lpm_ram_dq</code>	<code>lpm_ram_io</code>	<code>lpm_or</code>
<code>lpm_rom</code>	<code>lpm_shiftreg</code>	<code>lpm_xor</code>	<code>scfifo</code>

Intel FPGA DDR I/O Inference

The synthesis software maps Intel FPGA 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. 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;

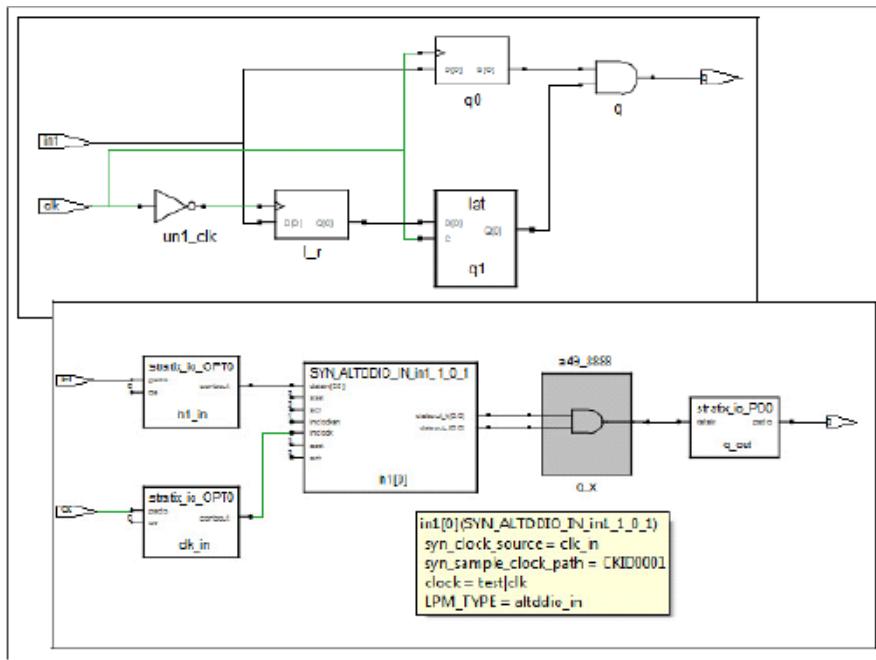
always @ (posedge clk)
begin
    q0 <= in1;
end
always @ (negedge clk)
begin
    l_r <= in1;
end

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 registers are triggered from the posedge of the clock.

```

--#####
-- The test case infers altddio_out instance for Intel FPGA.
--#####
--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

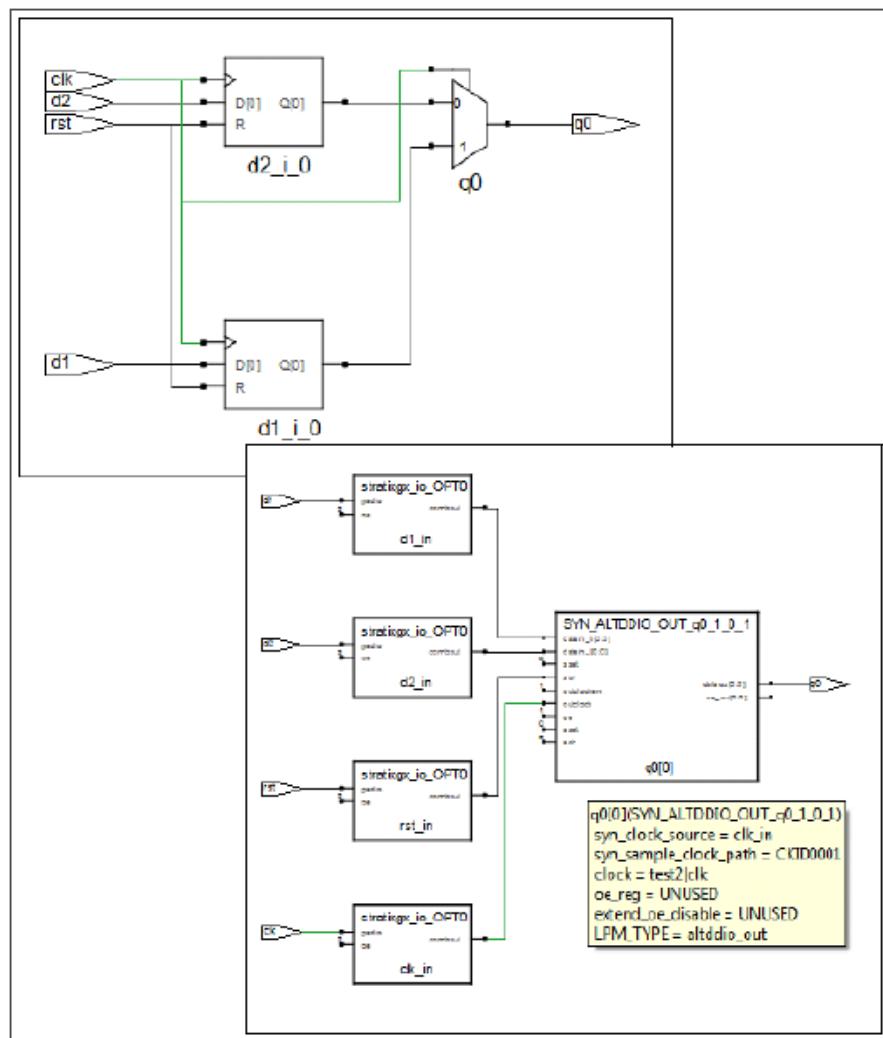
-----without any control-----
    if (clk'event and clk='1') then
        if rst ='1' then
            d1_i_0 <= '0';
        else
            d1_i_0 <= d1;
        end if ;
    end if;

    if (clk'event and clk='1') then
        if rst = '1' then
            d2_i_0 <= '0';
        else
            d2_i_0 <= d2;
        end if ;
    end if;

end process;

q0 <= d1_i_0 when clk ='1'  else d2_i_0;
-----
end behavioral;
```

The RTL and Technology views below show how `altdio_out` inferencing is implemented after synthesis.



Intel FPGA Constraints, Attributes, and Options

Specify timing constraints, general attributes, and Intel FPGA-specific attributes to improve your design. Manage these files with the SCOPE constraints and attributes editor. This section explains how to implement Intel FPGA constraints, attributes, and options. Refer to:

- [Intel FPGA I/O Standards](#), on page 599
- [Register Packing in Intel FPGA Designs](#), on page 614
- [Multi-dimensional Array Support in VQM](#), on page 615

Intel FPGA I/O Standards

Intel FPGA has vendor-specific I/O standard constraints it supports for synthesis. The following tables show the applicable I/O standards for the following families with their equivalent Quartus I/O standards:

- [Stratix and Stratix GX I/O Standards](#)
- [Stratix II and Stratix II GX I/O Standards](#)
- [Stratix III, IV, and V I/O Standards](#)
- [Cyclone I/O Standards](#)
- [Cyclone 10 GX and Cyclone 10 LP I/O Standard](#)
- [Arria 10 and Max 10 I/O Standards](#)

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

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix	Stratix GX
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
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

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix	Stratix GX
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

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

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix II	Stratix II 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	
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
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, IV, and V I/O Standards

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV	Stratix V
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	X

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV	Stratix V
DIFF_HSTL_12_II	Differential 1.2-V HSTL Class II	X	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	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	X
DIFF_HSTL_18_Class_II	Differential 1.8-V HSTL Class II	X	X	X
DIFF_HSUL_12	Differential 1.2-V HSUL			X
DIFF_SSTL_125	Differential 1.25-V SSTL			X
DIFF_SSTL_12	Differential 1.2-V SSTL			X
DIFF_SSTL_135	Differential 1.35-V SSTL			X
DIFF_SSTL_15_I	Differential 1.5-V SSTL Class I		X	X
DIFF_SSTL_15	Differential 1.5-V SSTL			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		

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV	Stratix V
DIFF_SSTL_18_Class_I	Differential 1.8-V SSTL Class I	X	X	X
DIFF_SSTL_18_Class_II	Differential 1.8-V SSTL Class II	X	X	X
DIFF_SSTL_2_Class_I	Differential 2.5-V SSTL Class I	X	X	X
DIFF_SSTL_2_Class_II	Differential 2.5-V SSTL Class II	X	X	X
HCSL	HCSL		X	X
HSTL_12_I	1.2-V HSTL Class I	X	X	X
HSTL_12_II	1.2-V HSTL Class II	X	X	X
HSTL_15_Class_I	1.5-V HSTL Class I	X	X	X
HSTL_15_Class_II	1.5-V HSTL Class II	X	X	X
HSTL_18_Class_I	1.8-V HSTL Class I	X	X	X
HSTL_18_Class_II	1.8-V HSTL Class II	X	X	X
HSUL_12	1.2-V HSUL			X
LVCMOS	3.0-V LVCMOS	X		
LVCMOS_12	1.2 V		X	X
LVCMOS_15	1.5 V		X	X
LVCMOS_18	1.5 V		X	X
LVCMOS_25	2.5 V		X	X
LVCMOS_33	3.3-V LVCMOSV		X	X
LVDS	LVDS		X	X
LVDS_E_1R	LVDS_E_R1	X	X	

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV	Stratix V
LVDS_E_3R	LVDS_E_3R	X	X	X
LVPECL	Differential LVPECL		X	X
LVTTL	3.0-V LVTTL	X		
LVTTL_33	3.3-V LVTTL		X	X
MINI_LVDS	mini-LVDS		X	X
MINI_LVDS_1R	mini-LVDS_1R	X	X	
MINI_LVDS_3R	mini-LVDS_3R	X	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	X
PCML_14	1.4-V PCML		X	X
PCML_15	1.5-V PCML		X	X
PCML_25	2.5-V PCML		X	X
RSDS	RSDS		X	X
RSDS_E_1R	RSDS_E_1R	X	X	
RSDS_E_3R	RSDS_E_3R	X	X	X
SSTL_125	SSTL-125			X
SSTL_12	SSTL-12			X
SSTL_135	SSTL-135			X
SSTL_15	SSTL-15			X
SSTL_15_I	SSTL-15 Class I	X	X	X
SSTL_15_II	SSTL-15 Class II	X	X	X
SSTL_18_Class_I	SSTL-18 Class I	X	X	X

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV	Stratix V
SSTL_18_Class_II	SSTL-18 Class II	X	X	X
SSTL_2_Class_I	SSTL-2 Class I	X	X	X
SSTL_2_Class_II	SSTL-2 Class II	X	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
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

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone	Cyclone II
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	

Cyclone 10 GX and Cyclone 10 LP I/O Standard

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone 10 GX	Cyclone 10 LP
BUS_LVDS	Bus LVDS		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_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

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone 10 GX	Cyclone 10 LP
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_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 1.2-V SSTL Class I		X
DIFF_SSTL_2_Class_II	Differential 1.2-V SSTL Class II		X
DIFF_HSUL_12	Differential 1.2-V HSUL	X	
DIFF_SSTL_125	Differential 1.25-V SSTL	X	
DIFF_SSTL_125_I	Differential 1.25-V SSTL	X	
DIFF_SSTL_125_II	Differential 1.25-V SSTL	X	
DIFF_SSTL_12	Differential 1.2-V SSTL	X	
DIFF_SSTL_12_I	Differential 1.2-V SSTL Class I	X	
DIFF_SSTL_12_II	Differential 1.2-V SSTL Class II	X	
DIFF_SSTL_135	Differential 1.35-V SSTL	X	
DIFF_SSTL_135_I	Differential 1.35-V SSTL Class I	X	
DIFF_SSTL_135_II	Differential 1.35-V SSTL Class II	X	

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone 10 GX	Cyclone 10 LP
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	Differential 1.5-V SSTL	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 2.5-V SSTL Class I		X
DIFF_SSTL_2_Class_II	Differential 2.5-V SSTL Class II		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
HSTL_18_Class_II	1.8-V HSTL Class II	X	X
HSUL_12	1.2-V HSUL	X	
LVCMOS_12	1.2V	X	X
LVCMOS_15	1.5V	X	X
LVCMOS_18	1.8V	X	X
LVCMOS_25	2.5V	X	X
LVCMOS_33	3.3-V LVCMOS		X
LVCMOS	3.0-V LVCMOS	X	
LVDS_E_3R	LVDS_E_3R		X
LVDS	LVDS	X	X

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone 10 GX	Cyclone 10 LP
LVPECL	Differential LVPECL	X	X
LVTTL	3.0-V LVTTL	X	X
LVTTL_33	3.3-V LVTTL		X
MINI_LVDS	mini-LVDS	X	X
MINI_LVDS_3R	mini-LVDS_E_3R		X
PCI	3.0-V PCI		X
PCI-X	3.0-V PCI-X		X
PPDS	PPDS		X
PPDS_E_3R	PPDS_E_3R		X
RSDS	RSDS	X	X
RSDS_E_1R	RSDS_E_1R		X
RSDS_E_3R	RSDS_E_3R		X
SSTL_125	SSTL-125	X	
SSTL_125_I	SSTL-125 Class I	X	
SSTL_125_II	SSTL-125 Class II	X	
SSTL_12	SSTL-12	X	
SSTL_12_I	SSTL-12 Class I	X	
SSTL_12_II	SSTL-12 Class II	X	
SSTL_135	SSTL-135	X	
SSTL_135_I	SSTL-135 Class I	X	
SSTL_135_II	SSTL-135 Class II	X	
SSTL_15	SSTL-15	X	
SSTL_15_Class_I	SSTL-15 Class I	X	
SSTL_15_Class_II	SSTL-15 Class II	X	
SSTL_18_Class_I	SSTL-18 Class I	X	X
SSTL_18_Class_II	SSTL-18 Class II	X	X

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone 10 GX	Cyclone 10 LP
Differential_12_V POD	Differential 1.2-V POD	X	
Current_Mode_Logic_(CML)	Current Mode Logic (CML)	X	
12_V POD	1.2-V POD	X	
High_Speed_Differential_I/O	High Speed Differential I/O	X	
SSTL_2_Class_I	SSTL-2 Class I		X
SSTL_2_Class_II	SSTL-2 Class II		X
SSTL_3_Class_I	SSTL-3 Class I		
SSTL_3_Class_II	SSTL-3 Class II		

Arria 10 and Max 10 I/O Standards

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Arria 10	Max 10
BUS_LVDS	Bus LVDS		
DIFF_HSTL_12_Class_I	Differential 1.2-V HSTL Class I	X	X
DIFF_HSTL_12_Class_II	Differential 1.2-V HSTL Class II	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	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	X
DIFF_HSUL_12	Differential 1.2-V HSUL	X	X
DIFF_SSTL_125	Differential 1.25-V SSTL	X	

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Arria 10	Max 10
DIFF_SSTL_12	Differential 1.2-V SSTL	X	
DIFF_SSTL_135	Differential 1.35-V SSTL	X	X
DIFF_SSTL_15	Differential 1.5-V SSTL Class I	X	X
DIFF_SSTL_15_I	Differential 1.5-V SSTL Class I	X	X
DIFF_SSTL_15_II	Differential 1.5-V SSTL Class II	X	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
HSTL_18_Class_II	1.8-V HSTL Class II	X	X
HSUL_12	1.2-V HSUL	X	X
LVCMOS	3.0-V LVCMOS	X	X
LVCMOS_12	1.2V	X	X
LVCMOS_15	1.5V	X	X
LVCMOS_18	1.8V	X	X
LVCMOS_25	2.5V	X	X

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Arria 10	Max 10
LVCMOS_33	3.3-V LVCMOS	X	X
LVDS_E_3R	LVDS_E_3R		X
LVDS	LVDS	X	X
LVPECL	Differential LVPECL	X	X
LVTTL_33	3.3-V LVTTL	X	X
LVTTL	3.0-V LVTTL	X	X
MINI_LVDS	mini-LVDS	X	X
MINI_LVDS_3R	mini-LVDS_E_3R		X
PCI	3.0-V PCI		X
PPDS	PPDS		X
PPDS_E_3R	PPDS_E_3R		X
RSDS_E_1R	RSDS_E_1R		X
RSDS_E_3R	RSDS_E_3R		X
RSDS	RSDS	X	X
SSTL_2_Class_I	SSTL-2 Class I		X
SSTL_2_Class_II	SSTL-2 Class II		X
SSTL_18_Class_I	SSTL-18 Class I	X	X
SSTL_18_Class_II	SSTL-18 Class II	X	X
SSTL_125	SSTL-125	X	
SSTL_12	SSTL-12	X	
SSTL_135	SSTL-135	X	X
SSTL_15	SSTL-15	X	X
SSTL_15_I	SSTL-15 Class I	X	X
SSTL_15_II	SSTL-15 Class II	X	X
Differential_12_V POD	Differential 1.2-V POD	X	

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Arria 10	Max 10
Current_Mode_Logic_ (CML)	Current Mode Logic (CML)	X	
12_V POD	1.2-V POD	X	
High_Speed_Differential _I/O	High Speed Differential I/O	X	
SLVS	SLVS	X	
Sub_LVDS	Sub-LVDS	X	
HiSpi	HiSpi	X	
TMDS	TMDS	X	
18_V_Schmitt_Trigger	1.8 V Schmitt_Trigger	X	
15_V_Schmitt_Trigger	1.5 V Schmitt_Trigger	X	
25_V_Schmitt_Trigger	2.5 V Schmitt_Trigger	X	
33_V_Schmitt_Trigger	3.3 V Schmitt_Trigger	X	

See Also:

- [Industry I/O Standards, on page 321](#) for a list of industry I/O standards.
- See [Intel FPGA Attribute and Directive Summary, on page 660](#) for a list of Intel FPGA attributes and directives.

Register Packing in Intel FPGA 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 Intel FPGA Designs, on page 643](#) for the procedure.

The `syn_useioff` attribute is supported in the compile point flow.

Multi-dimensional Array Support in VQM

To support multi-dimensional arrays and retain the port structures, add the option `support_multidim_vqm` to the top-level module of the project file. You cannot set this option in the GUI. This option will not preserve the multi-dimensional array port structure on the sub-modules in design hierarchy. See [Project Options for set_option/get_option](#), on page 150

Consider the following RTL example:

```
module test
(
    input clk1,
    input [3:0] in,
    input in2,
    output [3:0]out [1:0]
);
```

After synthesis, the array structure is reflected in the synthesis output netlist (.vqm):

<code>support_multidim_vqm</code> is not set (default)	<code>support_multidim_vqm</code> is set to 1
--	---

	<code>set_option -support_multidim_vqm 1</code>
<code>module test (</code> <code>clk1,</code> <code>in,</code> <code>in2,</code> <code>out</code> <code>)</code> <code>;</code>	<code>module test (</code> <code>clk1,</code> <code>in,</code> <code>in2,</code> <code>out</code> <code>)</code> <code>;</code>
<code>input clk1 ;</code> <code>input [3:0] in ;</code> <code>input in2 ;</code> <code>output [7:0] out ;</code>	<code>input clk1 ;</code> <code>input [3:0] in ;</code> <code>input in2 ;</code> <code>output [3:0]out[1:0];</code>

Intel FPGA Device Mapping Options

This section describes optimizations that are specific to Intel FPGA.

- [Core Voltage](#), on page 616
- [Companion Parts](#), on page 616
- [I/O Insertion in Intel FPGA Designs](#), on page 616
- [Sequential Optimizations in Intel FPGA Designs](#), on page 617
- [Fanout Limits in Intel FPGA Designs](#), on page 617
- [Compile Point Remapping in Intel FPGA Designs](#), on page 619

See Also

- [Stratix, Arria, and Cyclone Families](#), on page 621
- [MAX II, MAX V, MAX 10 Families](#), on page 624
- [MAX Family](#), on page 628
- [set_option Command for Intel FPGA Architectures](#), on page 631

Core Voltage

Certain Stratix III parts

You can specify a core voltage value for some parts. See [Specifying HardCopy and Stratix Companion Parts](#), on page 644 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 Core Voltage in Stratix III Designs](#), on page 645 for information on specifying companion parts.

I/O Insertion in Intel FPGA 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.

Sequential Optimizations in Intel FPGA Designs

The Disable Sequential Optimization option determines whether sequential optimizations are performed. By default, sequential optimizations are performed. The 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, which effectively disables the FSM Compiler and FSM Explorer. The Tcl equivalent for this option is `-no_sequential_opt 1|0`.

Fanout Limits in Intel FPGA 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 617](#)), and the `syn_maxfan` attribute ([The `syn_maxfan` attribute for Intel FPGA Designs, on page 618](#)).

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 Intel FPGA 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.
- 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).

Compile Point Remapping in Intel FPGA 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 626](#) and [Compile Point Basics, on page 606](#) 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 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 <i>not</i> 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 <i>interface</i> 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 <i>updated</i> 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: <ul style="list-style-type: none">• The compile point is remapped, and the Update Compile Point Timing Data option is enabled.• The interface of the compile point is changed.

Stratix, Arria, and Cyclone Families

This section provides guidelines for using the synthesis tool with Stratix (and corresponding HardCopy), MAX 10, Arria, and Cyclone devices.

- [Stratix, Arria, and Cyclone Device Mapping Options](#), on page 621
- [Stratix, Arria, and Cyclone File Format Options](#), on page 624
- [Intel FPGA Attribute and Directive Summary](#), on page 660

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.

This table alphabetically lists the options you can set on the Device tab:

Option	Description
Intel FPGA Models	<p>Specifies if Clearbox models are used for instantiated and inferred Intel FPGA megafunctions.</p> <ul style="list-style-type: none"> • <code>on</code> calls Clearbox for inferred and instantiated Megafunctions. • <code>off</code> does not call Clearbox for inferred and instantiated Megafunctions. • <code>clearbox_only</code> calls Clearbox for inferred Megafunctions only. <p>For more information, see Implementing Megafunctions with Clearbox Models, on page 688.</p>
Annotated Properties for Analyst	<p>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 <code>Find</code> command. See Annotating Timing Information in the Schematic Views, on page 475 in the <i>User Guide</i> for more information.</p>
Automatic Read/Write Check Insertion for RAM	<p>Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509.</p>
Companion Part	<p>Lets you specify a HardCopy companion part. See Specifying HardCopy and Stratix Companion Parts, on page 644.</p>

Option	Description
Core Voltage	<i>Stratix III</i> Lets you specify a core voltage. See Specifying Core Voltage in Stratix III Designs , on page 645 .
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 Intel FPGA Designs , on page 616 for details.
Disable Sequential Optimizations	Enables or disables the sequential optimizations for the design. See Sequential Optimizations in Intel FPGA Designs , on page 617 .
Exploratory Place and Route	<i>Synplify Premier</i> Runs different place-and-route configurations for the design that can favorably affect routability of the design.
Fanout Guide	Guide for fanout-based optimizations which results in buffering or replication of the net driver. See Setting Fanout Limits, on page 577 in the <i>User Guide</i> for more information. The default is 500.
Maximum Parallel Jobs for Exploratory Place and Route	<i>Synplify Premier</i> Specifies the number of CPUs available to use for parallel processing. Each parallel job uses one CPU. The default is 4.
MIF File Directories	Specifies the absolute directory path location for the missing MIF files. See Intel FPGA MIF Files , on page 567 for details.
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.
Synthesis Strategy	<i>Synplify Premier</i> Specifies the synthesis strategy to use for the design. Synthesis modes include: <ul style="list-style-type: none"> • advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but may require longer runtimes compared to other modes. This is the default. • base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes. • fast – Reduces the number of optimizations performed to improve the synthesis runtime. • routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.

Option	Description
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 Intel FPGA Designs, on page 619 for details.
Use TimeQuest Timing Analyzer	<i>Stratix II and Stratix II GX</i> 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 Constraints, on page 655 for additional information.

set_option Command 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 Intel FPGA Architectures, on page 631](#).

set_option Device Options

This is the syntax for setting device options for Stratix, MAX 10, 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
-fix_gated_and_generated_clocks 1|0
-maxfan value
-max_parallel_par_explorer value
-mif_files_dirs mifDirectoryPath
-no_sequential_opt 1|0
-par_explorer 1|0
-resolve_multiple_driver 1|0
-run_prop_extract 1|0
-rw_check_on_ram 1|0
```

```
-syn_altera_model on|off|clearbox_only  
-synthesis_strategy advanced|base|fast|routability  
-timequest 1|0  
-update_models_cp 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  
  -pipe 1|0  
  -resource_sharing 1|0  
  -retiming 1|0  
  -symbolic_fsm_compiler 1|0 or -autosm 1|0  
  -use_fsm_explorer 1|0  
  -validate_mif_files 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 Intel FPGA Architectures](#), on page 636 for details of the arguments.

MAX II, MAX V, MAX 10 Families

This section provides guidelines for using the synthesis tool with MAX II, MAX V devices. See the following:

- [Device Mapping Options for MAX II and MAX V](#), on page 625
- [MAX II and MAX V File Format Options](#), on page 628

- [Intel FPGA Attribute and Directive Summary](#), on page 660

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.

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 use them to create collections using the Tcl <code>Find</code> command. See Annotating Timing Information in the Schematic Views , on page 475 in the <i>User Guide</i> for more information.
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle , on page 509.
Clock Conversion	Performs gated and generated clock optimization when enabled. See Working with Gated Clocks , on page 828 and Optimizing Generated Clocks , on page 881 in the <i>User Guide</i> for details.
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 Intel FPGA Designs , on page 616.
Disable Sequential Optimizations	Enables or disables the sequential optimizations for the design. See Sequential Optimizations in Intel FPGA Designs , on page 617.
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 577 in the <i>User Guide</i> for more information.

Option	Description
Map Logic	Maps logic directly to Intel FPGA ATOM primitives. See Intel FPGA ATOM Primitives , on page 555 for more information. By default, this is enabled.
MIF File Directories	Specifies the absolute directory path location for the missing MIF files. See Intel FPGA MIF Files , on page 567 for details.
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.
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.
Synthesis Strategy	<p><i>Synplify Premier</i></p> <p>Specifies the synthesis strategy to use for the design. Synthesis modes include:</p> <ul style="list-style-type: none"> advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but may require longer runtimes compared to other modes. This is the default. base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes. fast – Reduces the number of optimizations performed to improve the synthesis runtime. routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.
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 Intel FPGA Designs , on page 619 , for an explanation.

set_option Command 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.w [set_option Command for Intel FPGA Architectures, on page 631](#).

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
  -mif_files_dirs mifDirectoryPath
  -no_sequential_opt 1|0
  -resolve_multiple_driver 1|0
  -run_prop_extract 1|0
  -rw_check_on_ram 1|0
  -synthesis_strategy advanced|base|fast|routability
  -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
  -validate_mif_files 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:

- | | |
|----------------|--|
| User Interface | <ul style="list-style-type: none"> • 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 Intel FPGA Architectures , on page 636 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 Intel FPGA MAX, on page 628](#)
- [File Format Options for MAX, on page 631](#)
- [Intel FPGA Attribute and Directive Summary, on page 660](#)

Device Mapping Options for Intel FPGA MAX

You can specify device mapping options in the Implementation Options dialog box or with the Tcl `set_option` command.

The Intel FPGA 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 use them to create collections using the Tcl Find command. See Annotating Timing Information in the Schematic Views, on page 475 in the <i>User Guide</i> for more information.

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
Disable Sequential Optimizations	Enables or disables the sequential optimizations for the design. See Sequential Optimizations in Intel FPGA Designs , on page 617 .
Map logic	Controls mapping to Intel FPGA 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.
MIF File Directories	Specifies the absolute directory path location for the missing MIF files. See Intel FPGA MIF Files , on page 567 for details.
Percent of designs to optimize for timing	Specifies the percentage of nets to optimize during synthesis. See Tips for Optimization, on page 554 of the <i>User Guide</i> .
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.
Soft LCELLS	Controls the insertion of soft buffers during synthesis.
Synthesis Strategy	<p><i>Synplify Premier</i></p> <p>Specifies the synthesis strategy to use for the design. Synthesis modes include:</p> <ul style="list-style-type: none"> • advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but may require longer runtimes compared to other modes. This is the default. • base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes. • fast – Reduces the number of optimizations performed to improve the synthesis runtime. • routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.

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 Intel FPGA Architectures, on page 631](#).

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 Intel FPGA Architectures, on page 631](#).

```
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
-mif_files_dirs mifDirectoryPath
-no_sequential_opt 1|0
-resolve_multiple_driver 1|0
-run_prop_extract 1|0
-rw_check_on_ram 1|0
-soft_buffers 1|0
-synthesis_strategy advanced|base|fast|routability
```

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
-validate_mif_files 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 | <ul style="list-style-type: none"> • 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 Intel FPGA Architectures , on page 636 for details of the arguments. |

set_option Command for Intel FPGA 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 149](#), or enter `help set_option` in a Tcl Script window

The following table describes the subset of options available for different Intel FPGA 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 636 for a list of valid keywords.
-part partName	Specifies a part for the implementation. Refer to Project->Implementation Options->Device or to the Intel FPGA 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 Intel FPGA documentation for available choices.

OPTION	DESCRIPTION
-package packageName	Specifies the package. Refer to Project-> Implementation Options->Device or to the Intel FPGA documentation for available choices.
Optimization Options	
-area_delay_percent value	<p><i>MAX technologies</i></p> <p>Specifies the percentage of nets to optimize during synthesis. Use only when <code>map_logic</code> is true. See Tips for Optimization, on page 554 of the <i>User Guide</i>.</p> <p>Default: 0</p>
-automatic_compile_point 1 0	<p><i>Stratix, Arria, and Cyclone technologies</i></p> <p>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.</p>
-block 1 0	<i>All technologies except MAX</i>
-disable_io_insertion 1 0	<p>Enables/disables I/O insertion during synthesis. See I/O Insertion in Intel FPGA Designs , on page 616.</p> <p>Default: 0 (Inserts I/Os)</p>
-clique 1 0	<i>MAX II and MAX V technology</i>
-cliqing 1 0	<p>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.</p> <p>Default: 1 (enabled)</p>
-domap 1 0	<p><i>MAX family</i></p> <p>Turns on direct mapping to LCELLS during synthesis.</p> <p>Default: 1 (enabled).</p>
-fanin_limit value	<p><i>MAX technology</i></p> <p>Sets the maximum fanin during synthesis. Use only when <code>map_logic</code> is true. To improve routability, reduce the value.</p> <p>Default: 40</p>

OPTION	DESCRIPTION
-fix_gated_and_generated_clocks 1 0	<i>All families except MAX, MAX II, and MAX V.</i> Performs gated and generated clock optimization when enabled. See Working with Gated Clocks, on page 828 and Optimizing Generated Clocks, on page 881 in the <i>User Guide</i> for details.
-map_logic 1 0	<i>MAX family</i> Turns on direct mapping to ATOMs or LCELLS during synthesis. See Intel FPGA ATOM Primitives, on page 555 for more information. Default: 1 (direct mapping enabled)
-maxfan value	<i>All families except MAX</i> Establishes a limit for fanout-based optimizations See Fanout Limits in Intel FPGA Designs, on page 617 for more information.
-max_parallel_par_explorer value	<i>Synplify Premier</i> Specifies the number of CPUs available to use for parallel processing. Each parallel job uses one CPU. The default is 4.
-mif_files_dirs mifDirectoryPath	Specifies the absolute directory path location for the missing MIF files. See Intel FPGA MIF Files, on page 567 for details.
-no_sequential_opt 1 0	Enables or disables the sequential optimizations for the design. See Sequential Optimizations in Intel FPGA Designs, on page 617 for details. Default: 0 (sequential optimizations enabled)
-par_explorer 1 0	<i>Synplify Premier</i> Runs different place-and-route configurations for the design that can favorably affect routability of the design.
-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 559 in the <i>User Guide</i> 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.

OPTION	DESCRIPTION
-resource_sharing 1 0	Enables or disables resource sharing. See Sharing Resources, on page 581 in the <i>User Guide</i> 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 563 in the <i>User Guide</i> 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 475 in the <i>User Guide</i> for more information.
-rw_check_on_ram 1 0	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For details about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
-soft_buffers 1 0	<i>MAX technology</i> Controls the insertion of soft buffers during synthesis. Buffers are inserted when the value is true. Default: true
-symbolic_fsm_compiler 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 585 in the <i>User Guide</i> for details of its use. Default: 0 or false (FSM synthesis disabled)
-autosm 1 0	
-syn_altera_model on off clearbox_only	Determines whether Clearbox is called for Intel FPGA Megafunctions during synthesis. For more information, see Implementing Megafunctions with Clearbox Models, on page 688 .

OPTION	DESCRIPTION
-synthesis_strategy advanced base fast routability	<p><i>Synplify Premier</i></p> <p>Specifies the synthesis strategy to use for the design. Synthesis modes include:</p> <ul style="list-style-type: none"> advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but may require longer runtimes compared to other modes. This is the default. base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes. fast – Reduces the number of optimizations performed to improve the synthesis runtime. routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.
-timequest 1 0	<p><i>Stratix II and Stratix II GX</i></p> <p>Determines whether to use the Intel Quartus II TimeQuest Timing Analyzer (1) or the Classic Timing Analyzer (0).</p> <p>The Quartus II TimeQuest Timing Analyzer is the default for Stratix IV, Stratix III, Arria GX, and Arria II GX devices.</p>
-update_models_cp 1 0	<p><i>All technologies except MAX</i></p> <p>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 Intel FPGA Designs, on page 619, for details.</p>
-use_fsm_explorer 1 0	<p>Enables or disables the FSM Explorer. See Running the FSM Explorer, on page 589 in the <i>User Guide</i> for information about using it.</p>
-validate_mif_files 1 0	<p>When enabled, checks that MIF files exist for associated Intel FPGA MegaWizard generated RAM or ROM instances. If MIF files are missing, the synthesis tool generates an error. See Intel FPGA MIF Files, on page 567 for more information.</p>
-voltage value	<p><i>Stratix III</i></p> <p>Sets a core voltage for Stratix III devices that support speed grade 4. See Core Voltage, on page 616.</p>

Technology Keywords

The following table lists the `-technology` keyword for different Intel FPGA technologies:

Family	Valid <code>-technology</code> Keywords
Arria	ARRIA-GX, ARRIAI-GX, ARRIAI-GZ, ARRIAV, ARRIA10
Cyclone	CYCLONE, CYCLONEII, CYCLONEIII, CYCLONEIV-E, CYCLONEIV-GX, CYCLONEV, CYCLONE10-GX, CYCLONE10-LP
MAX 10	MAX10
MAX	MAX3000
MAX II	MAXII, MAXV
Stratix	STRATIX, STRATIXGX, STRATIX_HC, STRATIXII, STRATIXII-GX, STRATIXIII, STRATIXIV, STRATIXV, HARDCOPYII, HARDCOPYIII, HARDCOPYIV

project Command for Intel FPGA 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 Intel FPGA devices. For a complete list of project command options, see [project](#), on page 111, or type `help project` in a `Tcl` Script window:

Option	Description
<code>-result_format vqm</code>	Specifies the synthesis result file format for the implementation. The format must be <code>vqm</code> (lower-case letters).
<code>-result_file filename</code>	Specifies the name of the synthesized netlist for the implementation.

Integration with Intel FPGA Tools and Flows

Use the following Intel FPGA tools or flows for optimizing your design:

- [Intel FPGA New Mapper](#), on page 637
- [Clearbox Support](#), on page 638
- [Greybox Support](#), on page 640
- [Working with Intel FPGA PLLs](#), on page 640
- [Instantiating Special Buffers as Black Boxes in Intel FPGA Designs](#), on page 641
- [Specifying Intel FPGA I/O Locations](#), on page 643
- [Packing I/O Cell Registers in Intel FPGA Designs](#), on page 643
- [Specifying HardCopy and Stratix Companion Parts](#), on page 644
- [Specifying Core Voltage in Stratix III Designs](#), on page 645
- [Using LPMs in Simulation Flows](#), on page 646

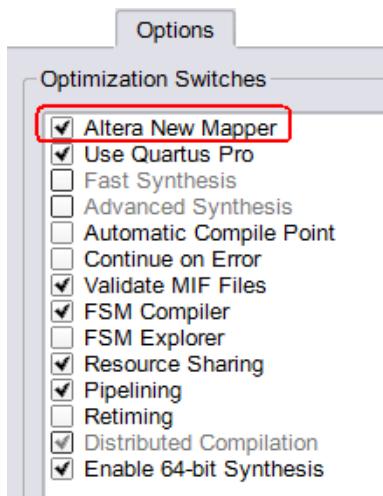
Intel FPGA New Mapper

Intel Arria 10, Arria V GZ, and Stratix V Technologies

The new Intel FPGA mapper provides performance and runtime improvements for the design. Older Intel FPGA devices must use the previous mapper; therefore both mappers will coexist.

To use the new Intel FPGA mapper, either:

- Enable the Intel FPGA New Mapper switch from the Options tab of the Implementation Options panel.



- Use the following Tcl command:

```
set_option -use_new_altera_mapper 1
```

Clearbox Support

Synplify Pro, 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.

Intel FPGA 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 and has known primitives like dffe or latch in addition to the WYSIWYG Clearbox primitives (see [Clearbox WYSIWYG Cells, on page 639](#)). 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 Intel FPGA technologies*.
- For some supported Intel FPGA 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, Stratix V, Stratix 10 and Agilex.

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 688](#) 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

Greybox Support

Synplify Pro, Synplify Premier

The synthesis tools support Intel FPGA 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 Intel FPGA 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 [Instantiating Megafunctions Using Grey Box Netlists, on page 699](#) in the *User Guide*.

Working with Intel FPGA PLLs

The synthesis software recognizes the Intel FPGA PLL component, `altpll`, from the Stratix, Cyclone, and Arria GX device families. The following procedure shows you how to use this component in your designs. The procedure uses the Intel FPGA Megafunction wizard to generate structural VHDL or Verilog files for the Intel FPGA PLLs.

1. If you are using VHDL, the `altpll` component normally will be declared in the MegaWizard file, and you can comment out the `LIBRARY` and `USE` clauses in the file. The following shows an example of the lines to be commented out:

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

If the component declaration in the MegaWizard file is not compatible with a particular Quartus software version, use the appropriate `vhd` file packaged with the Synopsys software in the corresponding `lib/altera/quartus_llnn` directory. For example, the `altera_mf.vhd` file for use with Quartus 10.1 is in the `quartus_ll101` subdirectory.

2. If you are using Verilog, no action is necessary as the mapper understands the `altpll` component.

For compatibility with different Quartus versions, `altera_mf.v` files are packaged with the software in the `lib/altera/quartus_llnn` directory. Use the

file from the directory that corresponds to the Quartus version that you are using.

3. Instantiate the altpll component in your design.
4. Add the MegaWizard Verilog or VHDL files to your project.
5. Open SCOPE and define the PLL input frequency in the SCOPE window. The synthesis software does not use the input frequency from the Intel FPGA MegaWizard software. Based on the input value you supply, the software generates the PLL outputs. All PLL outputs are assigned to the same clock group.
6. Set the target technology and the Quartus version (Implementation Options->Implementation Results), and synthesize as usual. The software uses the altpll component information and the constraints when synthesizing your design. The synthesis software forward-annotates the PLL input constraints to Quartus.

Instantiating Special Buffers as Black Boxes in Intel FPGA Designs

You can instantiate special buffers, like global buffers for clocks, sets/resets, and other heavily loaded signals, as black boxes in your design. See the Intel FPGA documentation for details about special buffers and the number of resources available for the part you are using.

1. Define a black-box module for your special buffer with the `syn_black_box` directive.

See the examples below and [syn_black_box, on page 149](#) in the *Attribute Reference Manual* for syntax details.

2. Use this black-box module to buffer the signals you want assigned to special buffers.
3. Synthesize the design and place-and-route as usual.

The Intel FPGA tools accept the black box.

Verilog Example of Instantiating Special Buffers as Black Boxes

```
module global(a_out, a_in) /* synthesis syn_black_box */ ;
  output a_out;
  input a_in;

  /* This continuous assignment is used for simulation,
   * but is ignored by synthesis. */
  assign a_out = a_in;
endmodule

module top(clk, pad_clk) ;
  output pad_clk;
  input clk;
  // pad_clk is the primary input
  global clk_buf(pad_clk, clk);
endmodule
```

VHDL Example of Instantiating Special Buffers as Black Boxes

```
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;

entity top is
  port (clk : out std_logic;
        pad_clk : in std_logic);
end top;

architecture structural of top is
  -- In this example, "global" is an Intel FPGA vendor macro directly
  -- instantiated in the Intel FPGA VHDL design as a black box.

  component global
    port(a_out : out std_logic; a_in : in std_logic) ;
  end component;

  -- Set the syn_black_box attribute on global to true.
  attribute syn_black_box of global: component is true;

  -- Declare clk, the internal global clock signal
begin
  -- pad_clk is the primary input
  clk_buf: global port map (clk, pad_clk);
end structural;
```

Specifying Intel FPGA I/O Locations

You can specify I/O locations in Intel FPGA designs using the `syn_loc` attribute. If you do not specify I/O locations, the P&R tool automatically assigns them locations.

1. If you used the QSF2SDC utility, to translate Intel FPGA QSF `set_location_assignment` and `set_instance_assignment` constraints, do nothing.

The utility automatically assigns the `syn_loc` attribute to the pins with constraints.

2. To define an I/O location manually, use the following syntax:

Top-level sdc file	define_attribute {portName} syn_loc {pinNumbers}
--------------------	---

Verilog	object /* synthesis syn_loc = "pinNumbers" */
---------	--

VHDL	attribute syn_loc of object : objectType is "pinNumbers"
------	---

Packing I/O Cell Registers in Intel FPGA Designs

You can improve input or output path timing in designs by packing registers into I/O cells with the `syn_useioff` attribute.

1. To pack the registers globally, set `syn_useioff=1` on the top-level module or architecture. Specify the attribute in the source code, the SCOPE interface, or directly in the constraint file.

Format	Example
---------------	----------------

Verilog	module test(d, clk, q) /* synthesis syn_useioff=1 */;
---------	---

VHDL	architecture rtl of test is attribute syn_useioff : boolean; attribute syn_useioff of rtl : architecture is true;
------	---

Constraint file syntax	define_global_attribute syn_useioff 1
---------------------------	---------------------------------------

2. To set the attribute locally, set `syn_useioff=1` on a port.

Format	Example
Verilog	<pre>module test(d, clk, q); input [3:0] d; input clk; output [3:0] q /* synthesis syn_useioff=1 */; reg q;</pre>
VHDL	<pre>entity test is port (d : in std_logic_vector (3 downto 0); clk : in std_logic; q : out std_logic_vector (3 downto 0); attribute syn_useioff : boolean; attribute syn_useioff of q : signal is true; end test;</pre>
Constraint file syntax	<code>define_attribute {p:q[3:0]} syn_useioff 1</code>

3. Synthesize the design.

The order of precedence used when there are conflicts for packing is registers, followed by ports, and finally global.

If `syn_useioff` is enabled for Arria GX and Stratix families, registers are not packed into Multiply/Accumulate (MAC) blocks.

The `syn_useioff` attribute is supported in the compile point flow.

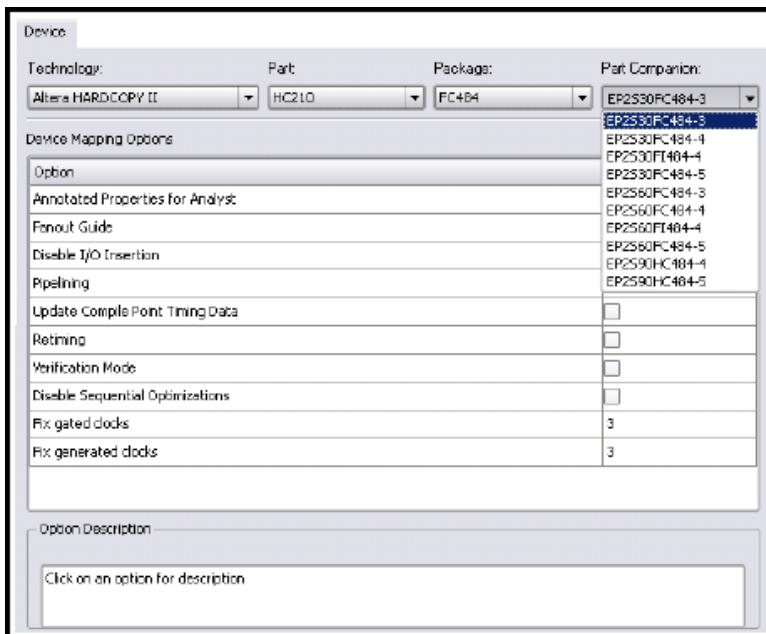
Specifying HardCopy and Stratix Companion Parts

For Stratix II, Stratix III, and Stratix IV devices, you can specify an associated HardCopy II, HardCopy III, or HardCopy IV companion part to allow you to migrate from Stratix to HardCopy in Quartus. By default, no companion part is specified for a Stratix device family. However, for a HardCopy device family, a Stratix companion part must be specified.

You select the companion device in the Device tab of the Implementation Options dialog box as in the following example:

You can use any of the following methods to specify companion parts:

- Select the companion device in the Device tab of the Implementation Options dialog box as shown here:



- Use this Tcl command, where *partName* is the part name and number:

```
set_option -part_companion partName
```

When you specify a companion part, the mapper targets the device with the least resources. For example, if your Stratix device has five memories and the companion HardCopy device has four memories, the mapper only uses four memory resources and maps the rest to logic.

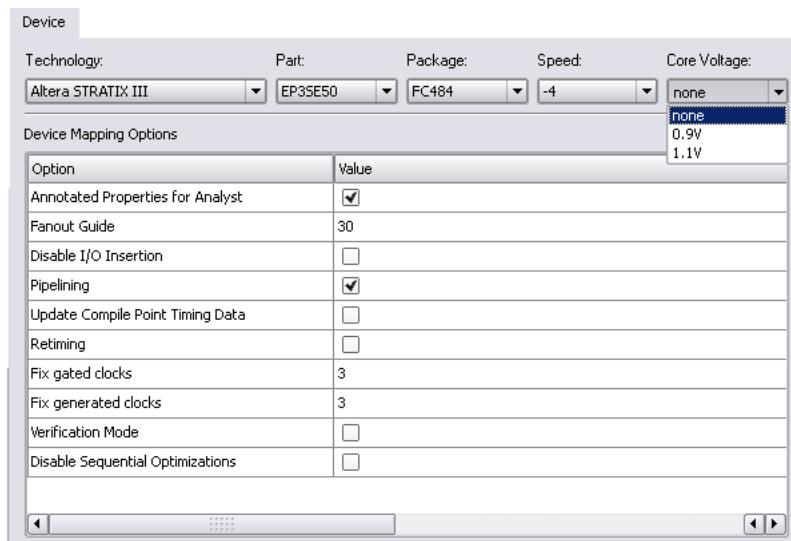
Specifying Core Voltage in Stratix III Designs

For some Stratix III devices, you can specify core voltage. Do the following:

- Click Implementation Options and do the following:
 - On the Device tab, set Technology to a Stratix III device.
 - Set Speed to -4.

This makes the Core Voltage option available.

- Set Core Voltage to the value you want, and click OK.



Alternatively, you can use the corresponding Tcl command: `set_option -voltage voltageValue`.

For example:

```
set_option -voltage 1.1V
set_option -voltage none
```

Using LPMs in Simulation Flows

This section describes how to use instantiated LPMs in simulation flows. For information about instantiating LPMs, see [Working with LPMs, on page 543](#).

Simulation Flows

The simulation flows vary, depending on the method used to instantiate the LPMs. For information about instantiating LPMs, see [Instantiating Intel FPGA LPMs Using VHDL Prepared Components, on page 548](#), [Instantiating Intel FPGA LPMs as Black Boxes, on page 544](#), and [Instantiating Intel FPGA LPMs Using a Verilog Library, on page 550](#). The following table summarizes the differences between the flows:

	Black Box Flow	Verilog Library/VHDL Prepared Component Flows
Applies to any LPM	Yes	No
Synthesis LPM timing support	No ¹	Yes
Synthesis procedure	Many steps	Simple
RTL simulation	Complicated steps	Easy
Post-synthesis (.vm) simulation	Yes	No
Post-P&R (.vo) simulation	Yes	Yes
Software version	Any version Max+PlusII Quartus II 1.0 or earlier	Quartus II 1.1 or later

1. Note: Black box timing models are not supported during synthesis. Therefore, there may be differences for maximum frequency between the synthesis and place-and-route timing reports.

Black Box Method Simulation Flow

Use the following flow when you instantiate LPMs as Verilog or VHDL black boxes. You can use this procedure for any LPM supported by Intel FPGA.

1. Use the Intel FPGA MegaWizard Plug-In Manager to create an LPM megafunction with the same module and port names as the black-box module in your synthesis design.
2. Compile the following:
 - Test bench
 - The design (RTL, post-synthesis `vm` file, or the post-P&R `vo` file)
 - The `v` file you generated in the previous step
3. Compile the LPM megafunction simulation model: `220model.v` or `altera_mf.v`.
4. For `vm` or `vo` simulation, compile the primitive simulation model.
5. Simulate the design.

Library/Prepared Component Simulation Flow

Use this simulation procedure when you use a Verilog library or VHDL prepared components to instantiate the LPMs. You can use this flow for `vo` simulation if your design contains the supported LPMs.

1. Instantiate the LPMs.
 - For VHDL designs, use the prepared components methods described in [Instantiating Intel FPGA LPMs Using VHDL Prepared Components, on page 548](#) or [Instantiating Intel FPGA LPMs as Black Boxes, on page 544](#).
 - For Verilog designs, use the library methods described in [Instantiating Intel FPGA LPMs Using a Verilog Library, on page 550](#) or [Instantiating Intel FPGA LPMs as Black Boxes, on page 544](#).
2. Compile the test bench and design. The design can be either RTL or the post-P&R `vo` file.
3. Compile the LPM megafunction simulation model: `220model.v` or `altera_mf.v`.
4. For `vo` simulation, compile the primitive simulation model. For example `apex20Ke_atoms.v`.
5. Simulate the design.

Intel FPGA Output Files and Forward Annotation

After synthesis, the software generates a log file and output files for Intel FPGA designs. The following describe some of the reports with Intel FPGA-specific information, or files that forward-annotate information to the Intel FPGA place-and-route tool. See the following topics:

- [Intel FPGA Reports](#), on page 649
- [Forward Annotating Intel FPGA Output](#), on page 652
- [Forward-annotation for Intel FPGA Constraints](#), on page 653

Intel FPGA 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 [Intel FPGA Custom Timing Reports](#), on page 652.

- [Intel FPGA Resource Usage Reports](#), on page 649
- [Intel FPGA Net Buffering Reports](#), on page 652
- [Intel FPGA Custom Timing Reports](#), on page 652

Intel FPGA Resource Usage Reports

The log file for Intel FPGA 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, Cyclone 10 LP
M20K	Agilex, Stratix 10, Stratix V, Arria 10, Cyclone 10 GX
M10K	Cyclone V, Arria V

- Number of I/Os
- Utilization percentage

See [Stratix II GX Resource Usage Report Example, on page 650](#) and [Cyclone III Resource Usage Report Example, on page 651](#) 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 #####

```

Intel FPGA 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 618](#).

Intel FPGA 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 482](#) 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.

Forward Annotating Intel FPGA Output

The following procedures show you how to pass information to the place-and-route tool; this information generally has no impact on synthesis. Typically, you use attributes to pass this information to the place-and-route tools. This section describes the following:

- [Specifying Pin Locations, on page 652](#)
- [Specifying Padtype and Port Information, on page 653](#)

Specifying Pin Locations

In certain technologies you can specify pin locations that are forward annotated to the corresponding place-and-route tool. The following procedure shows you how to specify the appropriate attributes.

1. Start with a design using an appropriate Intel FPGA technology.

2. Add the appropriate attribute to the port. For a bus, list all the bus pins, separated by commas.
 - To add the attribute from the SCOPE interface, click the Attributes tab and specify the appropriate attribute and value.
 - To add the attribute in the source files, use the appropriate attribute and syntax. For details about the attributes in the tables, see the *Attribute Reference Manual*.

Vendor Family	Attribute and Value
Intel FPGA	<code>syn_loc {pin_number}</code>

Specifying Padtype and Port Information

You can use an attribute to specify technology-specific port information or padtype.

Vendor	Attribute
Intel FPGA	<code>altera_io_powerup</code> <code>define_attribute {seg [31:0]} altera_io_powerup {high}</code>

Forward-annotation for Intel FPGA Constraints

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.

<code>create_clock</code>	Includes clock constraints generated by auto constraining.
<code>set_input_delay</code> <code>set_output_delay</code>	The synthesis tool forward-annotates the <code>set_input_delay</code> and <code>set_output_delay</code> constraints to Quartus II if you use the <code>syn_forward_io_constraints</code> 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 291 .
<code>set_multicycle_path</code>	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.

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.

Intel FPGA Max+Plus II Constraint Files

The constraint file generated for Intel FPGA Max+Plus II place-and-route tools has an .acf file extension. The following constraints are forward-annotated to Intel FPGA Max+Plus II in this file:

- `define_clock`
- `define_input_delay`
- `define_output_delay`

Forward-annotation Formats for Constraints

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. The software only forward-annotates the PLL inputs, because the back-end tool does its own propagation.

Forward-annotation of Intel FPGA 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.

Running Post-Synthesis Simulation with VHM Netlist

Intel FPGA supports language data types in a VHM netlist with the help of a wrapper file, `<results_file_name>_comp_top_wrapper.vhd`. After synthesis, the input RTL data types are not lost, but retained in the wrapper file. Use the wrapper file in which the data types are retained for post-synthesis simulation.

To generate the wrapper file:

1. Run synthesis as usual.
2. The tool generates a .vhd file and a wrapper file, `<results_file_name>_comp_top_wrapper.vhd`. This wrapper file is located in the directory, `implementationDirectory/synwork/`.

This file has the top-level entity from the VHM netlist instantiated in it. This wrapper file has same data types for the top-level signals in entity as the input RTL data types. Hence the input RTL data types are retained in the wrapper file.

3. Compile the wrapper file with the .vhd netlist to run simulation.

Limitations

The following data types are not supported in the wrapper file:

X01Z, UX01Z, X01, UX01, and enumerated data types.

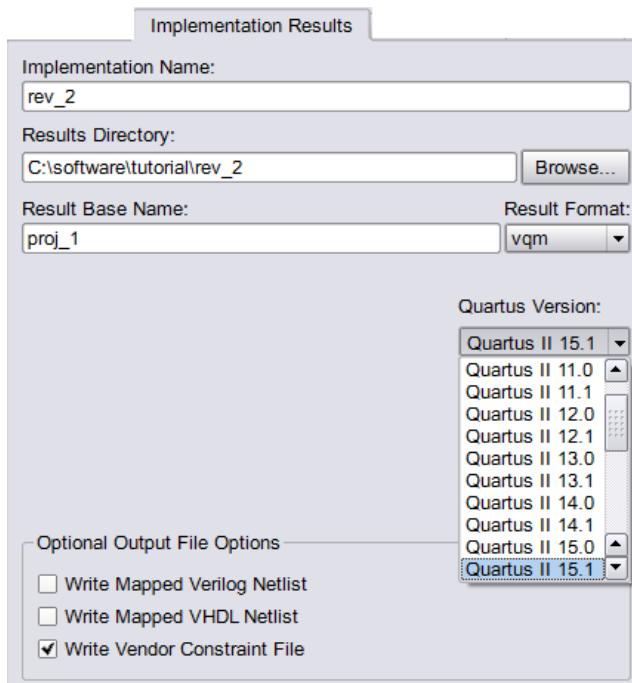
Intel FPGA Place-and-Route Guidelines

Here are some guidelines for running the Intel FPGA place-and-route tool. Refer to the following topics:

- [Run Intel FPGA Placement and Routing](#), on page 657
- [VQM Filenames for Place-and-Route Tool](#), on page 658
- [Quartus II Incremental Compilation Flow](#), on page 659

Run Intel FPGA Placement and Routing

The synthesis tools support many versions of the Intel FPGA place-and-route tools, and automatically generate the appropriate Intel FPGA 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.



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 Prime, on page 1106](#) and [Running P&R Automatically after Synthesis, on page 1102](#) of the *User Guide* for details.

Limitations for Intel FPGA Place and Route

Note the following limitation for Intel FPGA place and route.

Place and Route Runtime Degradation Unless Using Quartus II 14.1 on RHEL 6

Unless you run integrated place and route with Intel Quartus II 14.1 on Red Hat Enterprise Linux 6, you may experience a degradation in runtime.

Running Quartus II 14.1 on Linux 5 (or later versions) can cause this degradation. For more information, please contact Intel FPGA.

As a workaround, run integrated place and route with Quartus II 14.1 on Red Hat Enterprise Linux 6.

VQM Filenames for Place-and-Route Tool

Intel FPGA 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.

Quartus II Incremental Compilation Flow

Synplify Pro, Synplify Premier

The Intel 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 synthesis tool supports this Intel FPGA 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 Intel FPGA Quartus II Incrementally, on page 1109](#)
- [Synthesizing Compile Points, on page 626](#)

Intel FPGA Attribute and Directive Summary

The table below summarizes attributes available with Intel FPGA technology.

Attribute/Directive	Description
<code>altera_io_powerup</code>	Controls the power-up mode for I/O registers that do not have predefined preset or clear conditions.
<code>altera_logiclock_location</code>	Assigns a compile point to an Intel FPGA LogicLock region, specifying the location of the region).
<code>altera_logiclock_size</code>	Assigns a compile point to an Intel FPGA LogicLock region, specifying the size of the region.
<code>black_box_pad_pin</code> (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.
<code>black_box_tri_pins</code> (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.
<code>full_case</code> (D)	Specifies that a Verilog case statement has covered all possible cases.
<code>loop_limit</code> (D)	Specifies a loop iteration limit for for loops.
<code>parallel_case</code> (D)	Specifies a parallel, multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
<code>syn_allow_retiming</code>	Determines whether registers may be moved into combinational logic to improve performance.
<code>syn_allowed_resources</code>	Specifies allowed resources for compile points. The value assigned to a given compile point includes the resources used by its children (at all levels).
<code>syn_black_box</code> (D)	Defines a black box for synthesis.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Attribute/Directive	Description
<code>syn_direct_enable</code> (D and A)	Identifies the signal to use as the enable input to an enable flip-flop, when multiple candidates are possible.
<code>syn_edif_name_length</code>	Determines the length of port names in an Intel FPGA output EDIF file.
<code>syn_encoding</code>	Specifies the encoding style for state machines.
<code>syn_enum_encoding</code> (D)	Specifies the encoding style for enumerated types (VHDL only).
<code>syn_forward_io_constraints</code>	Enables I/O constraints to forward-annotate to the place-and-route tool.
<code>syn_force_seq_prim</code> (D)	Indicates that the fix gated clocks algorithm can be applied to the associated primitive.
<code>syn_gatedclk_clock_en</code> (D)	Specifies the name of the enable pin within a black box.
<code>syn_gatedclk_clock_en_polarity</code> (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.
<code>syn_hier</code>	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
<code>syn_highrel_iointerface</code>	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.
<code>syn_isclock</code> (D)	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
<code>syn_keep</code> (D)	Prevents the internal signal from being removed during synthesis and optimization.
<code>syn_loc</code>	Specifies pin locations for I/O pins and cores, and forward-annotates this information to the place-and-route tool.
<code>syn_maxfan</code>	Overrides the default fanout guide for an individual input port, net, or register output.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Attribute/Directive	Description
<code>syn_multstyle</code>	Determines whether multipliers are implemented in logic or hardware blocks.
<code>syn_netlist_hierarchy</code>	Determines whether the output netlist is flat or hierarchical.
<code>syn_noarrayports</code>	Prevents the ports in the output netlist from being grouped into arrays, and leaves them as individual signals.
<code>syn_noclockbuf</code>	Turns off automatic clock buffers for entire modules or nets, or just for specific input ports.
<code>syn_no_compile_point</code>	Ignores modules as compile points in the Automatic Compile Point flow.
<code>syn_noprune</code> (D)	Controls the automatic removal of instances with outputs that are not driven.
<code>syn_pipeline</code>	Specifies that registers be moved into multipliers to improve frequency.
<code>syn_preserve</code> (D)	Enables or prevents sequential optimizations, which eliminate redundant registers and registers with constant drivers.
<code>syn_probe</code>	Adds probe points for testing and debugging.
<code>syn_radhardlevel</code>	Enables the triple modular redundancy and voting logic in the design.
<code>syn_ramstyle</code>	Determines how RAMs are implemented.
<code>syn_reference_clock</code>	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
<code>syn_replicate</code>	Controls the replication of registers.
<code>syn_resources</code>	Specifies the resources available for black boxes. It is attached to Verilog black-box modules or VHDL architectures or component definitions.
<code>syn_romstyle</code>	Determines how ROM architectures are implemented.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Attribute/Directive	Description
<code>syn_rw_conflict_logic</code>	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.
<code>syn_safe_case</code>	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.
<code>syn_safefsm_pipe</code>	Removes the pipeline register on the error recovery path for the Preserve and Decode Unreachable States option.
<code>syn_sharing(D)</code>	Specifies resource sharing of operators.
<code>syn_shift_resetphase</code>	Allows you to remove the flip-flop on the inactive clock edge, built by the reset recovery logic for an FSM when a single event upset (SEU) fault occurs.
<code>syn_srlstyle</code>	Specifies register packing for Stratix altshift_tap mega function.
<code>syn_state_machine(D)</code>	Determines if the FSM Compiler extracts a structure as a state machine.
<code>syn_tco<n>(D)</code>	Defines the clock to output delay timing through a black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tpd<n> (D)</code>	Specifies timing propagation for combinational delay through a black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tristate(D)</code>	Specifies that a black box pin is a tristate pin.
<code>syn_tsu<n> (D)</code>	Specifies the timing setup delay for input pins, relative to the clock. The <i>n</i> indicates a value between 1 and 10.
<code>syn_useenables</code>	Prevents generation of registers with clock enable pins.
<code>syn_useioff</code>	Determines whether flip-flops are packed in the I/O ring to improve input/output path timing.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Attribute/Directive	Description
<code>syn_vote_loops</code>	Specifies whether or not to add any synchronous voter to a sequential feedback path for a distributed TMR module.
<code>syn_vote_register</code>	Enables voter logic to be inserted after every register of the module specified for distributed TMR.
<code>translate_off/translate_on</code> (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.

CHAPTER C

Designing with Lattice

This chapter discusses the following topics for synthesizing Lattice designs:

- [Basic Support for Lattice Designs](#), on page 666
- [Lattice Features](#), on page 671
- [Lattice Constraints, Attributes, and Options](#), on page 706
- [Lattice Device Mapping Options](#), on page 710
- [Lattice Design Options](#), on page 737
- [Lattice Output Files and Forward Annotation](#), on page 740
- [Integration with Lattice Tools and Flows](#), on page 744
- [Lattice Attribute and Directive Summary](#), on page 754

Basic Support for Lattice Designs

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 666
- [Netlist Format](#), on page 667

Supported Device Families

The synthesis tool creates technology-specific netlists for the following Lattice technologies:

FPGA	CPLD
ECP5UM5G/ECP5UM/ECP5U	ispGAL
iCE5LP	ispGDX/GDX2
iCE40UP, LIFCL, LFD2NX (Lattice Radiant software)	ispLSI1K, 2K 5K, 8K, ispLSI5000VE
iCE40UL/iCE40LM/iCE40/iCE40UP (iCEcube2 software)	ispLSI1K, 2K 5K, 8K, ispLSI5000VE
ispXPGA	ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG
LatticeECP5U/ECP5UM/ECP5UM5G	ispXPLD 5000MX
LatticeECP3	MACH
LatticeECP2S/ECP2MS/ECP2M/ECP2/ECP /EC	
LatticeSCM/SC	
LatticeXP2/LatticeXP	
LIFMD	
MachXO3LP/MachXO3L/MachXO2/MachX O	
ORCA FPSC, Series 2, Series3, Series 4	
Platform Manager 2, Platform Manager	

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 during the Lattice place-and-route. These files have an edf or edn extension. For Lattice Radiant devices, the tool outputs verilog netlist files for use during Lattice Radiant place-and-route. These files have a vm extension.

You can also use the project Tcl command to specify the result file format.

```
project -result_format edif
```

To generate Lattice output, see [Targeting Output for Lattice, on page 667](#).

Targeting Output for Lattice

You can generate output targeted for Lattice.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

The following table summarizes the outputs to set for the different devices, and shows the P&R tools for which the output is intended.

Vendor Support	Output Netlist	P&R Tool
Lattice (iCE40 UltraPlus and LIFCL)	vm(.vm)	Radiant Software
Lattice (newer device families)	EDIF (.edn)	Diamond
Lattice (iCE40 and iCE5 device families)	EDIF (.edf)	iCEcube2
Lattice Classic (ORCA families)	EDIF (.edn)	ispLEVER
Lattice Classic (Mach/isp device families)	EDIF (.edf)	ispExpert

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

Lattice 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 Diamond or ispLever place-and-route software. The Lattice constraint file uses the `lpf` extension. This constraint file must be imported into the Lattice flow.

For Lattice iCE families, 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.

For Lattice Radiant families, the software generates the output files (`vm` and `ldc`). The `ldc` constraint file forward annotates timing constraints for the Lattice Radiant 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).

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 [Specifying SCOPE Constraints, on page 210](#)).
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 can be forward-annotated to ispLEVER in this file:

- `set_false_path`
- `set_multicycle_path`
- `set_reg_input_delay`
- `set_reg_output_delay`
- `global frequency`

You can forward-annotate multicycle and false path constraints to the Lattice place-and-route tool by following the procedure below.

1. To forward-annotate a from, to, or through multicycle constraint, open the SCOPE spreadsheet and do either of the following:
 - Click the Multi-Cycle Paths tab. Depending on the type of constraint you want to set, select or type the instance name under the To, From or Through column. Next, set the number of clock cycles under the Cycles column.

When you set this constraint, the software runs timing-driven synthesis and then forward-annotates the constraint.

- Click the Other tab. In the Command column, type `define_multicycle_path`. In the Arguments column, type `-from` and the source port or register name, and `-to` and the destination port or register name. For example: `-from in0_int -to output 2`.

When you set this constraint from the Other tab, the software forward-annotates the constraint, but does not run timing-driven synthesis using this constraint.

2. To forward-annotate a false path constraint, open the SCOPE spreadsheet and do either of the following:
 - Click the False Paths panel. Depending on the type of constraint you want to set, select or type the instance name under the To, From or Through column. When you set this constraint, the software runs timing-driven synthesis and then forward-annotates the constraint.
 - Click the Other tab. In the Command column, type `define_false_path`. In the Arguments column, type `-from` and the source port or register name, and `-to` and the destination port or register name. For example:

Command	Arguments
<code>set_false_path</code>	<code>-from in1_int -to output</code>
<code>set_false_path</code>	<code>-from in* -to out*</code>

When you set this constraint from the Other tab, the software forward-annotes the constraint, but does not run timing-driven synthesis using this constraint.

3. Select Project->Implementation Options and enable the Write Vendor Constraint File option on the Implementation Results tab.
4. Run your design. The synthesis tool creates the `$DESIGN_synplify.lpf` file in the same directory as your results files.
5. Start the Lattice ispLEVER place-and-route tool and run the Map stage (after importing the `$DESIGN_synplify.lpf` file).
6. Run the PAR and BIT stages in ispLEVER.

Lattice Features

This section describes some supported Lattice components:

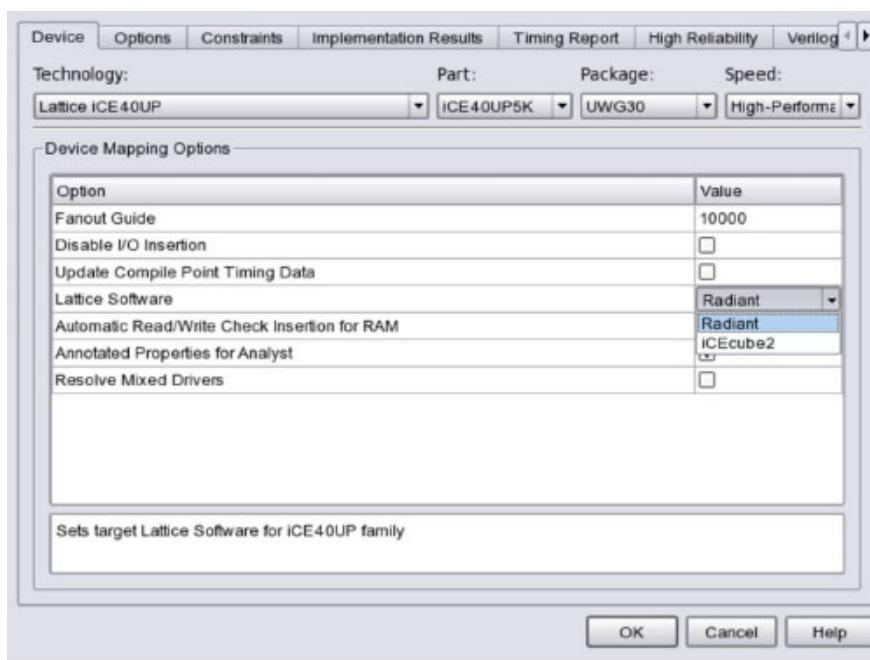
- [Lattice Radian Software Support](#), on page 671
- [Lattice Block RAM Support](#), on page 672
- [Lattice iCE RAM and ROM Support](#), on page 683
- [Lattice iCE Component Optimizations](#), on page 684
- [Initial Value Support for Registers](#), on page 687
- [Shift Chain Inference](#), on page 689
- [SB_MAC16 Block Inference](#), on page 690
- [Lattice Latch Support](#), on page 693
- [Timing Propagation Support for Lattice Oscillators](#), on page 696
- [Timing Propagation Through PLLs](#), on page 698
- [Timing Propagation Support for Instantiated Primitives](#), on page 702
- [Lattice Macros and Black Boxes](#), on page 703

Lattice Radian Software Support

The Lattice Radian software flow is supported on the Radian iCE40 UltraPlus device. There are two flows available in the Implementation Options, depending on the software you choose.

To choose:

1. Select Project->Implementation Options.
2. In the Device tab of the Implementation Options dialog box, choose Lattice iCE40UP as Technology.
3. In the Device Mapping Options, click Lattice Software and choose either Radiant or iCEcube2.



- If you select the iCEcube2 option, the mapper uses the iCEcube2 implementation and if you select the Radiant option, the mapper uses the Lattice Radiant implementation.

Lattice Block RAM Support

LatticeEC/ECP Technologies

- For the LIFCL device, the tool supports the inference of SP16K, PDP16K, PDPSC16K, DP16K, SPSP512K, PDPSC512K and DPSC512K RAM primitives.
- The three Block RAM primitives supported are:
 - 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.

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.

Single Address RAM Styles

RAMN_RW

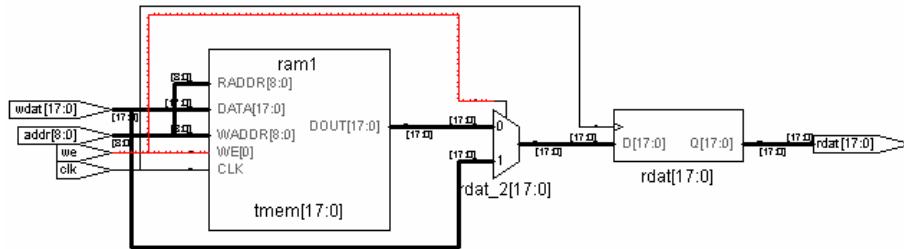
The read address and write address are the same with the exception of a flip-flop on the read data out. Based on the RTL coding style, WRITEFIRST, READFIRST, and NO_CHANGE modes apply. The mapper always uses the SP8KA Block RAM primitive.

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.

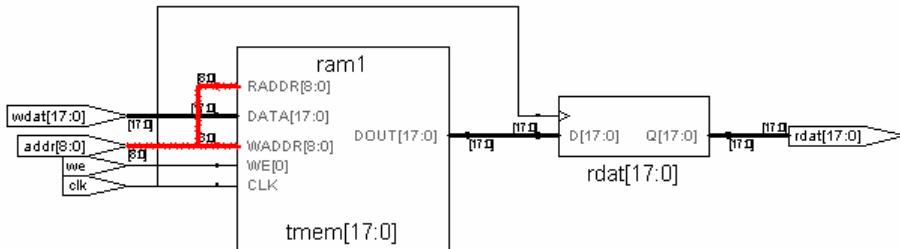
WRITE_FIRST Style

The Block RAM is set to WRITE_FIRST mode. Bypass logic is not required.



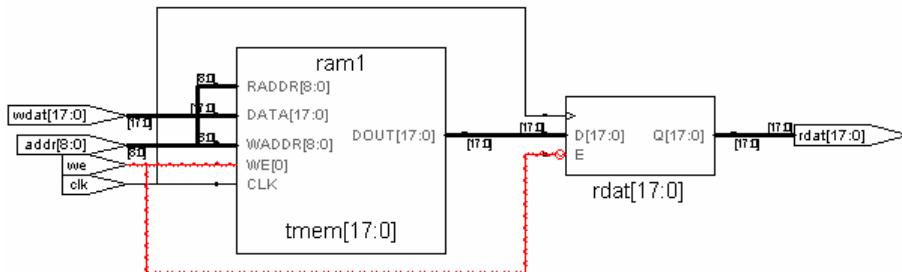
READ_FIRST Style

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.

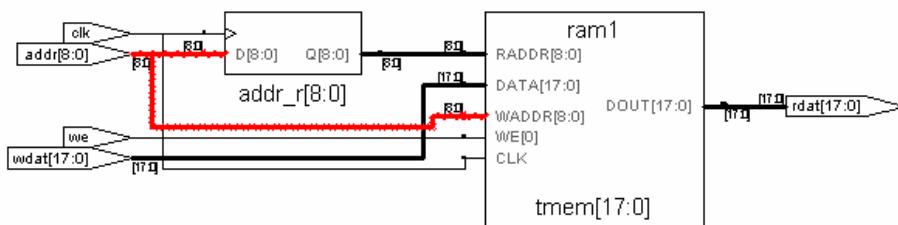


The ports are DOUT, ADDR, DIN, WE, CLK, OCLK, RST, EN

```
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.



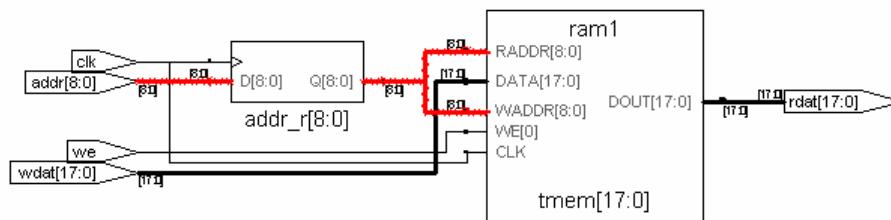
- **Synchronous Set/Reset:** If the address flip-flop has a set/reset signal, the RAM is considered as a generic case of RAM_R_W and mapped using dual-port RAMs.
- **Enable:** An address flip-flop with an enable cannot be mapped to the clock enable of the Block RAM primitive because it will affect the 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

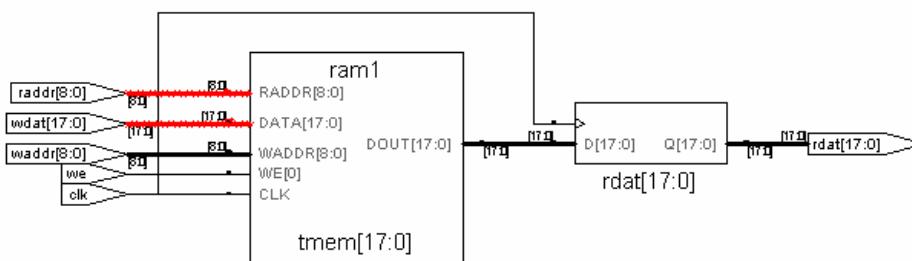
RAMN_R_W

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.



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'},
};
```

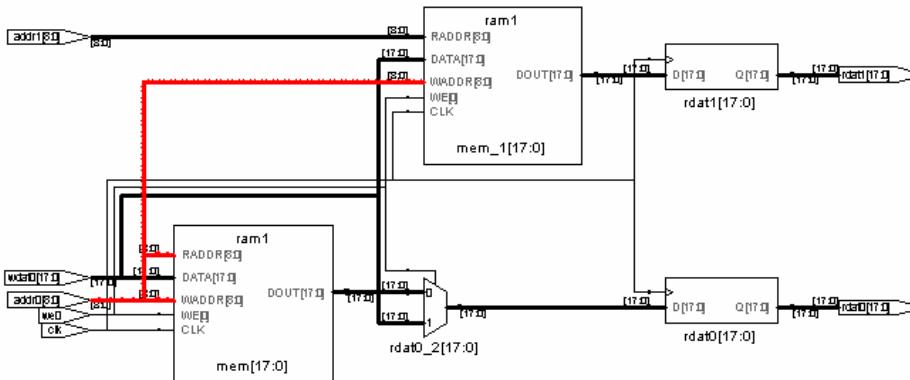
RAMN_RW_R

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.

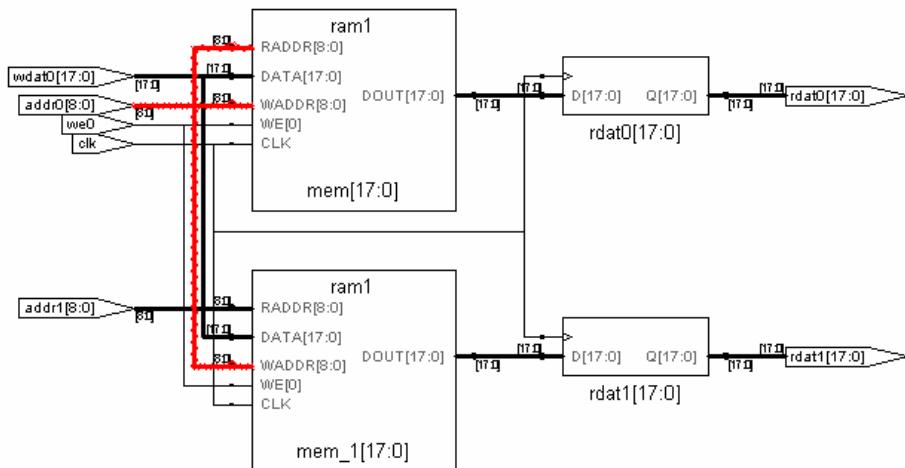
WRITE_FIRST Style

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.



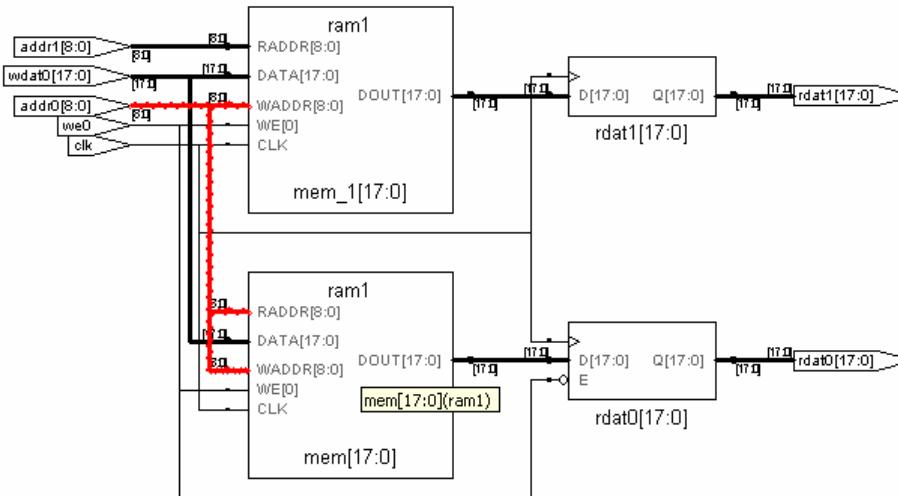
READ_FIRST Style

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.

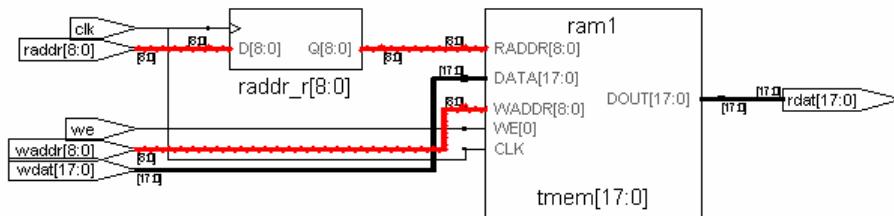


The ports are R_DOUT, RADDR, W_DOUT, DIN, WADDR, WE, CLK, R_OCLK, W_EN, EN, R_RST, W_RST

```
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.



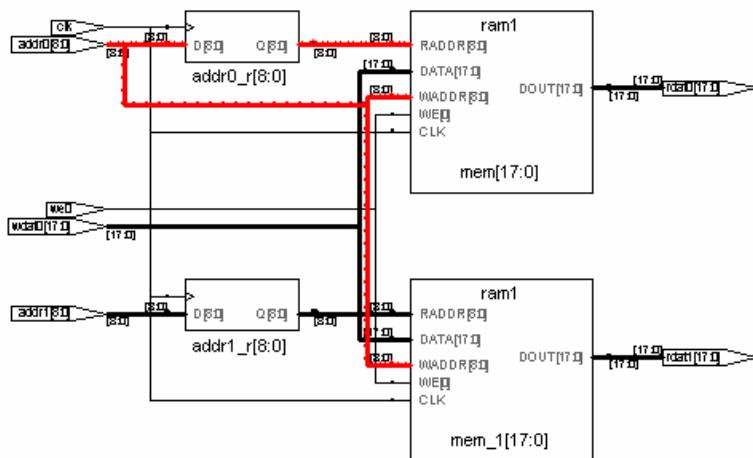
- **Synchronous Set/Reset:** is allowed on the address flip-flop. The read address to Block RAM is ANDed with reset or ORed with set. Because the write address to the Block RAM is fed from the output of the address flip-flop, the flip-flop can support set/reset.
- **Enable:** is allowed on the address flip-flop. Read address to Block RAM is modified.

The ports are DOUT, RADDR, DIN, WADDR, WE, CLK, OCLK

```
genericinfo r_w_generic[] = {
    {"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.



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 365](#) in the *Reference 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 703](#) in the *User Guide*. For general information about instantiating black boxes, see [Instantiating Black Boxes and I/Os in Verilog, on page 514](#) and [Instantiating Black Boxes and I/Os in VHDL, on page 516](#).

Programmable I/O Cell Latches

LatticeSC/SCM, LatticeXP/XP2, and LatticeEC/ECP families

The tool automatically infers programmable I/O cell (PIC) latches. Use `syn_keep` if you do not want to infer a PIC. See [Inferring Lattice PIC Latches, on page 744](#) in the *User Guide* for details and examples.

Lattice iCE RAM and ROM Support

Lattice iCE40 and ICE40LM Technologies

Single-port or dual-port synchronous memory are supported for these technologies.

RAM Mapping

Lattice 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 [Automatic RAM Inference, on page 229](#) of the *User Guide*.

Block RAMs can be configured as:

- Single-port memory
- Simple dual-port memory

Lattice 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 [Initializ-ing Values in RTL, on page 684](#).

Initializing Values in RTL

Initial values specified in the RTL for memories can be mapped to startup values on the FPGA. Lattice iCE40 devices support power on startup values for memories. See [Initializing RAMs](#), on page 528 in the *User Guide* for step-by-step procedures. You can initialize values in Verilog and VHDL. See:

- [Initializing RAMs in Verilog](#), on page 528
- [Initializing RAMs in VHDL](#), on page 529

Lattice iCE Component Optimizations

This section describes the following topics when working with Lattice iCE40 and ICE40LM designs:

- [Using Sequential Logic](#)
- [Using Combinational Logic](#)
- [Handling Tristates](#)
- [Handling I/Os and Buffers](#)

Using Sequential Logic

Lattice iCE40 and ICE40LM Technologies

The synthesis software infers or instantiates the following sequential logic depending on the control signal.

Primitive	Description
SB_DFF	D flip-flop
SB_DFFE	D flip-flop with clock enable
SB_DFFSR	D flip-flop with synchronous reset
SB_DFFR	D flip-flop with asynchronous reset
SB_DFFSS	D flip-flop with synchronous set
SB_DFFS	D flip-flop with asynchronous set
SB_DFFESR	D flip-flop with clock enable and synchronous reset
SB_DFFER	D flip-flop with clock enable and asynchronous reset
SB_DFFESS	D flip-flop with clock enable and synchronous set
SB_DFFES	D flip-flop with clock enable and asynchronous set
SB_DFFN	D flip-flop - negative edge clock
SB_DFFNE	D flip-flop - negative edge clock and clock enable
SB_DFFNSR	D flip-flop - negative edge clock with synchronous reset
SB_DFFNR	D flip-flop - negative edge clock with asynchronous reset
SB_DFFSS	D flip-flop - negative edge clock with synchronous set
SB_DFFNS	D flip-flop - negative edge clock with asynchronous set
SB_DFFNESR	D flip-flop - negative edge clock, enable and synchronous reset
SB_DFFNER	D flip-flop - negative edge clock, enable and asynchronous reset
SB_DFFNESS	D flip-flop - negative edge clock, enable and synchronous set
SB_DFFNES	D flip-flop - negative edge clock, enable and asynchronous set

Limitations

The synthesis software does not support initial values on flip-flops for the Lattice iCE technologies. The Power ON state for Lattice iCE40 flip-flops is 0.

Using Combinational Logic

Lattice iCE40 and ICE40LM Technologies

The synthesis software handles combinational logic as follows:

- This logic is mapped to the 4-input LUT (SB_LUT4).
- All data path operators (adders, subtractors, comparators, or multipliers) is mapped to carry chain logic (SB_CARRY) and additional LUT4s (SB_LUT4).

Handling Tristates

Lattice iCE40 and ICE40LM Technologies

The synthesis software handles tristates as follows:

- Internal tristates are not supported and are converted to MUXes whenever possible. An error is generated if an internal tristate cannot be converted to a MUX.
- Tristates connected to an output port are absorbed into the I/O pad.

Handling I/Os and Buffers

Lattice iCE40 and ICE40LM Technologies

The synthesis software handles I/Os and buffers as follows:

- The SB_IO primitive is used for inserting I/O pads.
- Since DDR inferencing is not supported, only input, output, and enable registers are packed into the I/O pads when available.
- The synthesis software does not infer SB_IO_DS, but allows for its instantiation.
- SB_GB is used for global buffers, such as clocks.
- SB_GB_IO is used for external clocks and SB_GB is used for internal clocks.

- Instantiated buffers are retained as is.
- Undriven pins in instantiated instances are left floating as it is in the RTL. Input pins in the RTL should not be connected to a floating net.
- The SB_WARMBOOT primitive can be instantiated and is treated as a black box.

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.

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 PLL (CLKIN, FB, MCLK, NCLK, LOCK, INTFB) /* synthesis
    syn_black_box */;
    input CLKIN, FB;
    output MCLK, NCLK, INTFB, LOCK;
    parameter VCOTAP = "1";
endmodule // EXPPLA

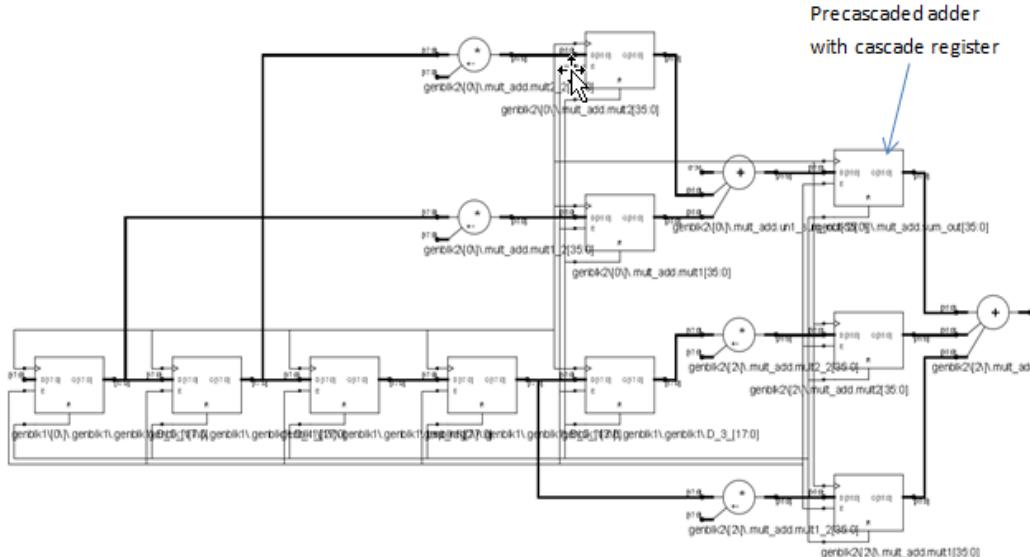
module macro (clki, clko);
    input clki;
    output clko;
    parameter VCOTAP = "1";
    defparam u1.VCOTAP = VCOTAP;
    PLL u1 (.CLKIN(clki), .FB(fb), .MCLK(clko), .NCLK(),
             .INTFB(fb), .LOCK());
endmodule // macro
```

```
module top (clk1i, clk1o1, clk1o2);
  input clk1i;
  output clk1o1, clk1o2;
  defparam u1.VCOTAP = "2";
  defparam u2.VCOTAP = "6";
  macro u1 (.clk1i(clk1i), .clk1o(clk1o1));
  macro u2 (.clk1i(clk1i), .clk1o(clk1o2));
endmodule // top
```

Shift Chain Inference

Lattice ECP3 Technologies

The following example shows a direct form FIR filter for a pre-cascaded adder structure with cascade registers that can automatically infer a shift chain.



Limitations

Packing shift chains for a direct form FIR filter:

1. Does not occur when the:
 - Adder tree structure does not have the proper cascade registers.
 - Delay tap registers are not in sequence in the adder tree.

2. Only applies for the MULT18X18 block. The MULT9X9 block is not supported.

SB_MAC16 Block Inference

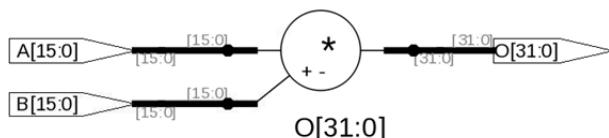
Lattice iCE5LP Technologies

The logic synthesis software infers SB_MAC16 primitives for better performance and resource utilization. The following structures in the HDL can be mapped to the SB_MAC16 block:

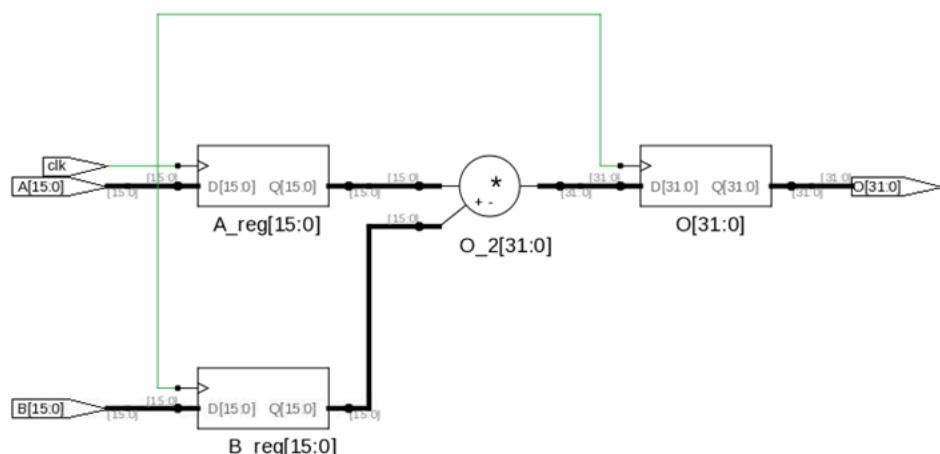
- Multipliers

Signed/unsigned multipliers with or without input and output registers are packed into the SB_MAC16 block for the following types of multiplier structures:

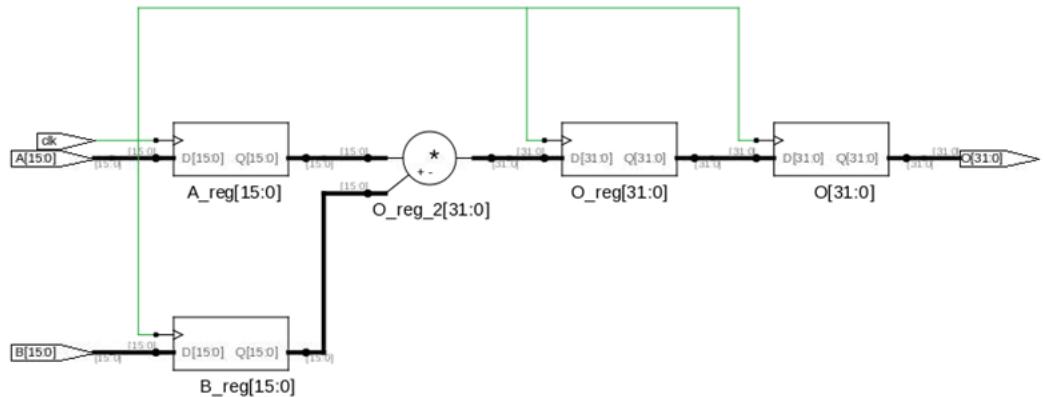
- Signed/unsigned multiplier in bypassed mode



- Signed/unsigned multiplier in intermediate register bypassed mode



- Signed/unsigned multiplier in all register pipeline mode

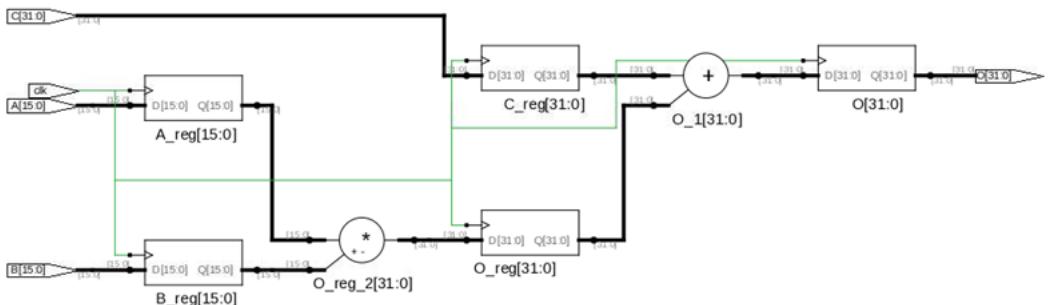


Note the following conditions:

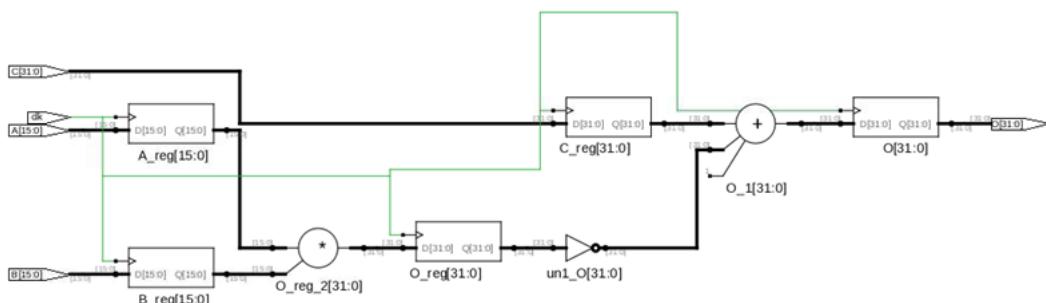
- No other structures except multipliers are packed in bypass mode.
- Multipliers with input widths greater than 16 are fractured into smaller sized multipliers. The tool can pack partial products in the SB_MAC16 block.
- Synthesis does not support multiply 8x8 mode because of hardware limitations. So one 8x8 multiplier consumes a full SB_MAC16 block.
- Mult-add/mult-sub (multiplier followed by an adder or subtractor)

Signed or unsigned multiplier-adder/subtractor can be packed into the SB_MAC16 block for the following mult-add/mult-sub structures:

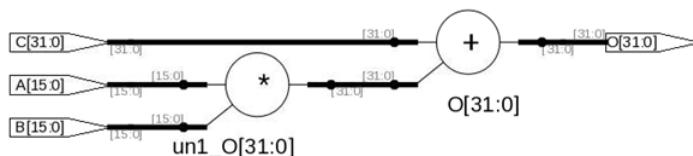
- Signed/unsigned multiplier-adder in intermediate register bypassed mode



- Signed/unsigned multiplier-subtractor in intermediate register bypassed mode



- Unsigned multiplier-adder/subtractor in bypassed mode



- Overflow bit logic mode

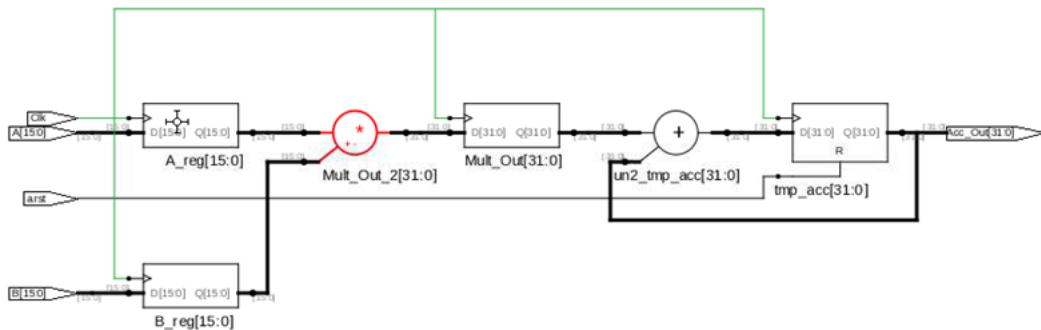
When the output width of the mult-add/sub is 33, the synthesis tool creates glue logic for the overflow bit that is bit number 33. If the width is greater than 33, then the adder is mapped to logic.

Note: Place-and-route limitations prevent signed mult-add/sub in bypass mode to be packed into the SB_MAC16 block. Only the multiplier is packed in bypass mode.

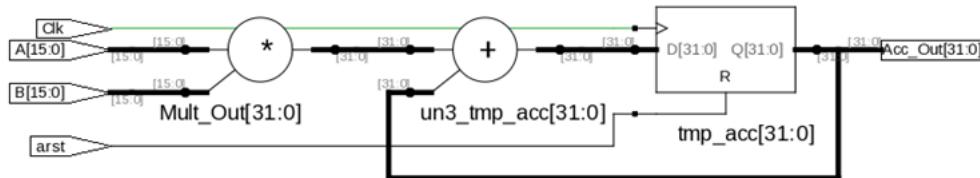
- MACC (Multiply Accumulate)

Signed/unsigned multiplier-accumulators are packed into the SB_MAC16 block for the following MACC structures:

- Signed or unsigned MAC_32 in intermediate register bypassed mode



- Unsigned MACC_32 in bypass mode



When the output width of the MACC is 33 or greater, the adder is mapped to logic.

Note: Place-and-route limitations prevent signed MACC in bypass mode to be packed into the SB_MAC16 block. Only the multiplier is packed in bypass mode.

Lattice Latch Support

The software supports the following types of latches:

- [Programmable I/O Cell Latches](#), on page 694
- [Lattice iCE Latches](#), on page 694

Programmable I/O Cell Latches

LatticeSC/SCM, XP/XP2, and EC/ECP Technologies

The tool automatically infers programmable I/O cell (PIC) latches. Use `syn_keep` if you do not want to infer a PIC. See [Inferring Lattice PIC Latches, on page 744](#) in the *User Guide* for details and examples.

Lattice iCE Latches

Lattice iCE40 and iCE40LM Technologies

The software implements latches using LUTs for the specified technology. Types of latches supported include:

- Simple latch
- Latch with asynchronous/synchronous set
- Latch with asynchronous/synchronous reset

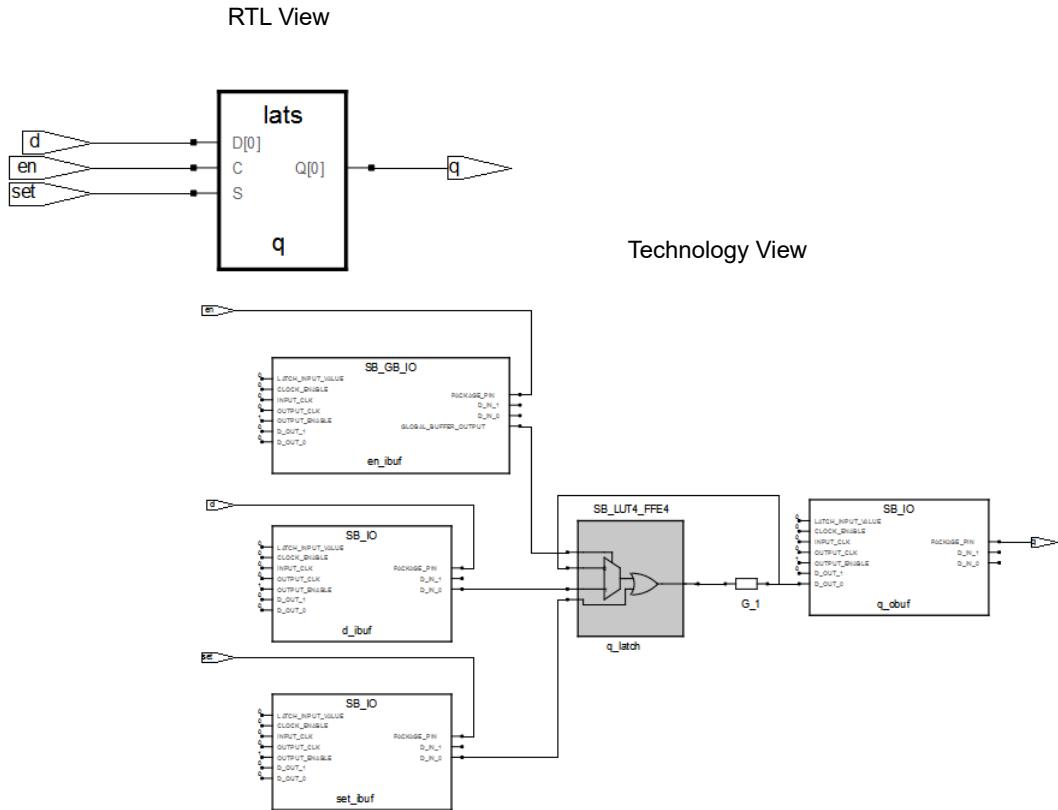
Example 1: Verilog RTL for a Latch with Asynchronous Set

```
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;
    reg q;

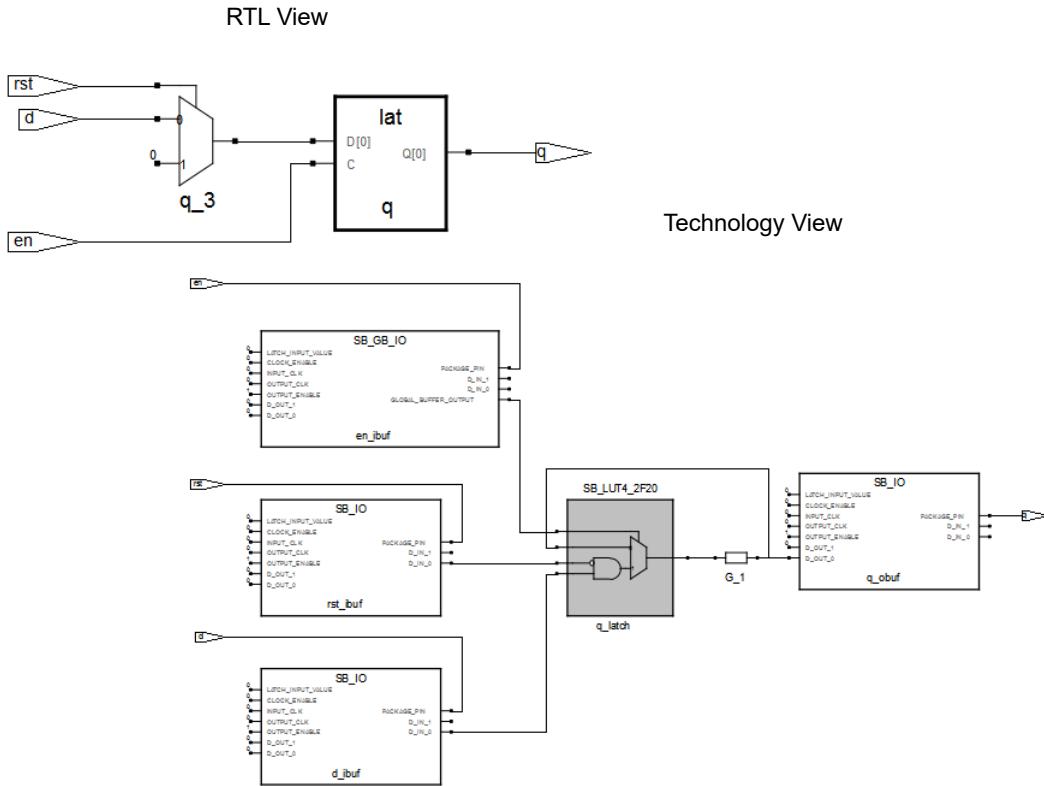
    always @ (en or d or rst)
    begin
  
```

```

if(en)
  if(rst) q <= 0;
  else q <= d;
end
endmodule

```

The following figure shows the schematic representations for a latch with synchronous reset.



Timing Propagation Support for Lattice Oscillators

Timing propagation supports Lattice Oscillator primitives in the following ways:

- The software computes frequency based on the parameter value of the instantiated oscillator primitive and is used for timing the paths.

- The calculated frequency is forward-annotated to the LPF file.
- If a user constraint is specified on the oscillator output, the constraint is honored and then forward-annotated to the LPF file.
- If no user constraint is specified on the oscillator output, instead of forward-annotating the constraint, the tool writes it as a comment (#FREQUENCY NET "clk" 2.5 MHz) in the forward-annotation file (lpf).
- The parameter value is written as a property in the output EDIF netlist.
- If an invalid parameter value is specified, the tool generates an error.

For the supported Lattice oscillator primitives, see the following table:

Oscillator Primitive	Supports the following devices ...
OSCD	ECP2/ECP2M
OSCF	ECP3
OSCG	ECP5U/ECP5UM
OSCE	XP2
OSCH	LPTM2/MachXO2/MachXO3L

RTL Usage Examples (OSCD)

Verilog RTL

```
OSCD inst0 (
    .CFGCLK(OSC_CLK)
);
defparam inst0.NOM_FREQ = "2.5";
```

VHDL RTL

```
COMPONENT OSCD
  GENERIC(NOM_FREQ: string:= "2.5");
  PORT (CFGCLK: OUT std_logic);
END COMPONENT;

...
Inst0 : OSCD
  GENERIC MAP (NOM_FREQ => "2.5")
  PORT MAP (CFGCLK => OSC_CLK);
```

LPF Output

```
#FREQUENCY NET "OSC_CLK" 2.5 MHz
```

The parameter value (NOM_FREQ) is written to the EDIF file. For the RTL examples above, the property is written in the edif output as shown below:

EDIF Output

```
(instance inst0 (viewRef verilog (cellRef OSCD))
  (property NOM_FREQ (string "2.5")))
```

Timing Propagation Through PLLs

Lattice iCE40 and ICE40LM Technologies

The synthesis tool can propagate timing constraints through instantiated PLLs for P-series devices of the Lattice technology.

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

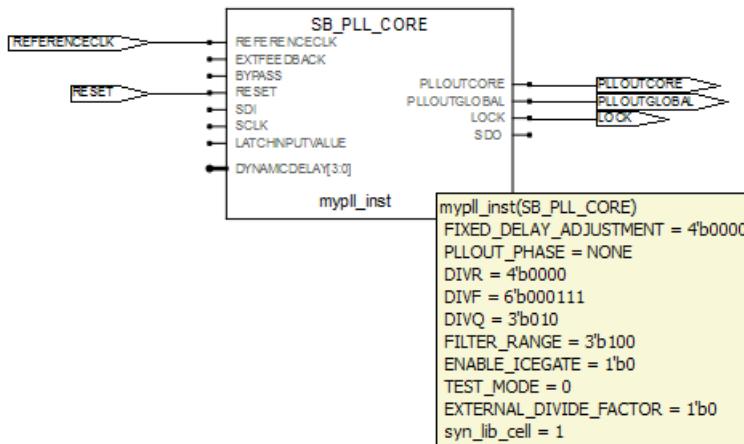
module test_pll
  (REFERENCECLK,
   PLLOUTCORE,
   PLLOUTGLOBAL,
   RESET,
   LOCK);

  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;

  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),

```

```

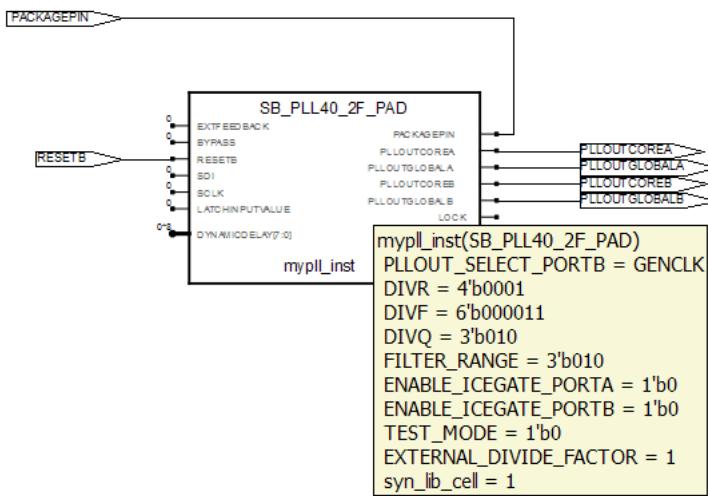
        .PLLOUTGLOBALA(PLLOUTGLOBALA),
        .PLLOUTGLOBALB(PLLOUTGLOBALB),
        .RESETB(RESETB),
        .BYPASS(1'b0));

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.



Timing Propagation Support for Instantiated Primitives

iCE40LM Technology

For the SB_LSOSC, SB_HSOSC, I2C, and SPI instantiated primitives, timing propagation enhancements include the following:

- The tool does not forward annotate inferred clocks based on the output of the primitives in the scf file.
- The software computes frequency based on the parameters that are used for timing the paths.

If a user constraint is specified on the output pin of the primitives, the constraint is forward annotated in the scf file. The software uses this user constraint, instead of the computed frequency for timing the paths.

RTL Usage Example (SB_I2C):

The following Verilog and VHDL code segments show how timing propagation can be specified for instantiated primitives.

Verilog RTL

```
SB_I2C Inst(
    .SBCLKI(sbclk),
    .SBRWI(sbrwi),
    .SBSTBI(sbstbi),
    ...
    .SDAOE(sdaoe)
) /* synthesis SB_I2C_MODE = "MASTER" I2C_CLK_DIVIDER = 2 */;
```

VHDL RTL

```
COMPONENT SB_I2C
  GENERIC (
    I2C_SLAVE_INIT_ADDR : String := "0b1111100001";
    BUS_ADDR74 : String := "0b0110");
  PORT (
    SBCLKI : in std_logic;
    SBRWI : in std_logic;
    ...
  );
```

```

);
END COMPONENT;
attribute SB_I2C_MODE :string;
attribute SB_I2C_MODE of inst_name : label is MASTER;
attribute I2C_CLK_DIVIDER : integer;
attribute I2C_CLK_DIVIDER of inst_name : label is 2;

```

The tool writes the divider factor (I2C_CLK_DIVIDER) as a property in the EDIF file as shown below:

```

(instance inst_name (viewRef PRIM (cellRef SB_I2C (libraryRef
    sb_ice)))
(property SB_I2C_MODE (string "MASTER"))
(property I2C_CLK_DIVIDER (integer 2))

```

Lattice 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 703](#).

Instantiating Lattice Macros

You can instantiate Lattice macros that are predefined in the Lattice libraries that come with the tool, in the *installDirectory/lib* directory.

1. To use a Verilog macro library, add the appropriate library to your project, making sure that it is the first file in the source files list.

The Verilog macro libraries are under the *installDirectory/lib* directory: Add the library appropriate to the technology and language (v or vhd) you are using

ORCA device families	<i>installDirectory/lib/lucent/orca*</i> Replace the asterisk with either 2, 3, or 4, according to the ORCA series you are using
LatticeXP device families	<i>installDirectory/lib/lucent/xp</i> <i>installDirectory/lib/lucent/xp2</i> <i>installDirectory/lib/lucent/xp3</i>
LatticeSC/SCM device families	<i>installDirectory/lib/lucent/sc</i> <i>installDirectory/lib/lucent/scm</i>
MachXO device families	<i>installDirectory/lib/lucent/machxo</i> <i>installDirectory/lib/lucent/machxo2</i>
ECP/EC device families	<i>installDirectory/lib/lucent/ec</i> <i>installDirectory/lib/lucent/ecp</i> <i>installDirectory/lib/lucent/ecp2</i> <i>installDirectory/lib/lucent/ecp3</i>
ispXPGA devices	<i>installDirectory/lib/lattice/lava1</i>
CPLD devices	<i>installDirectory/lib/cpld/lattice</i>

2. To use a VHDL macro library, add the appropriate `library` and `use` clauses to your VHDL source code at the beginning of the design units that instantiate the macros.

You only need the VHDL macro libraries for simulation, but it is good practice to add them to the code. The library names may vary, depending on the map file name, which is often user-defined. The simulator uses the map file names to point to a library.

CPLD devices	<code>library lattice;</code> <code>use lattice.components.all;</code>
ORCA device families	Replace the asterisk with the series number (2, 3, or 4) for the Lattice ORCA Series 2, Series 3, or Series 4 macro library you are using. <code>library orca*;</code> <code>use orca*.orcacomp.all;</code>

LatticeXP device families	library xp; use xp.components.all library xp2; use xp.components.all library xp3; use xp.components.all
LatticeSC/SCM device families	library sc; use sc.components.all library scm; use scm.components.all
MachXO device families	library machxo; use machxo.components.all library machxo2; use machxo.components.all
ECP/EC device families	library ec; use ec.components.all library ecp; use ecp.components.all; library ecp2; use ecp2.components.all; library ecp3; use ecp3.components.all;
ispXPGA device families	library lava; use lava.components.all;

-
3. Instantiate the macros from the library as described in [Instantiating Black Boxes and I/Os in Verilog, on page 514](#) and [Instantiating Black Boxes and I/Os in VHDL, on page 516](#).

Lattice Constraints, Attributes, and Options

The synthesis tools let you specify timing constraints, general HDL attributes, and Lattice-specific attributes to improve your design. You can manage the attributes and constraints in the SCOPE interface. This section explains how to implement Lattice constraints, attributes, and options. Refer to:

- [Lattice I/O Standards](#), on page 706
- [Disable I/O Insertion Globally or Port by Port](#), on page 707
- [I/O Buffer Support](#), on page 707
- [Global Buffer Promotion for Control Signals](#), on page 708
- [Hierarchical Attribute Support for Module Generation](#), on page 708
- [Inferring Carry Chains in Lattice XPLD Devices](#), on page 709
- [Selective Clock Enable Inference](#), on page 709

Lattice I/O Standards

Lattice has vendor-specific I/O standard constraints it supports for synthesis. The following table contains the applicable I/O standards for Lattice iCE40 devices.

Lattice 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

NOTE: The I/O standards apply for L04, L08, and P04 devices. The L01 device only supports SB_LVCMOS.

Lattice I/O Standards

SB_LVCMOS25_8	SB_LVDS_INPUT
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 321](#) for a list of industry I/O standards.
- [Chapter 4, Constraint Commands](#) for the timing constraints.
- [Lattice Attribute and Directive Summary, on page 754](#) 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 283](#) for more information.

I/O Buffer Support

LatticeECP/EC, SC/SCM, MACH, XP, and ORCA Technologies

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.

Global Buffer Promotion for Control Signals

Lattice iCE40 and iCE40LM Technologies

Control signals having fanout greater than the threshold values specified in the table below are inserted with global buffers automatically, if there are sufficient numbers of global buffers available on the device. The control nets with high fanout have priority.

Net	Global buffer inserted for the threshold
Asynchronous Set/Reset	16
Synchronous Set/Reset	16
Clock Enable	16

To override the 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 the `syn_insert_buffer="SB_GB"` attribute for nets or `syn_insert_buffer="SB_GB_IO"` attribute for ports so that the tool inserts a global buffer on the particular net/port, which has fanout less than the threshold value.

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 (clk_i, clk_o);
    input clk_i;
    output clk_o;
    parameter VCOTAP = "1";
    defparam u1.VCOTAP = VCOTAP;
    PPLL u1 (.CLKIN(clk_i), .FB(fb), .MCLK(clk_o), .NCLK(),
              .INTFB(fb), .LOCK());
    endmodule // macro

module top (clk_i, clk_o1, clk_o2);
    input clk_i;
    output clk_o1, clk_o2;
    defparam u1.VCOTAP = "2";
    defparam u2.VCOTAP = "6";
    macro u1 (.clk_i(clk_i), .clk_o(clk_o1));
    macro u2 (.clk_i(clk_i), .clk_o(clk_o2));
    endmodule // top

```

Inferring Carry Chains in Lattice XPLD Devices

For XPLD devices, you can control the inference of carry chains with the `syn_use_carry_chain` attribute. By default, all counters are implemented as carry chains when they are over 4 bits wide. To override this, set the `syn_use_carry_chain` attribute with a value of 0 on the registers of the counter or adder. For more information, see [Selective Carry-Chain Inferencing, on page 753](#).

Selective Clock Enable Inference

For CPLD devices, you can extract clock enable to enhance performance using the `syn_useenables` attribute. For more information, see [syn_useenables, on page 711](#).

Lattice 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

This section contains some options that appear on the Device tab for Lattice devices.

- [Fanout Guide](#), on page 711
- [Disable I/O Insertion](#), on page 711
- [GSR Resource](#), on page 711
- [Compile Point Remapping in Lattice Designs](#), on page 712
- [Write Declared Clocks Only](#), on page 713
- [Lattice Device Mapping Options \(Older Technologies\)](#), on page 714

See Also

- [ORCA Devices](#), on page 715
- [LatticeECP/EC and Later Devices](#), on page 717
- [LatticeSC/SCM and LatticeXP2/XP Devices](#), on page 719
- [MachXO and Platform Devices](#), on page 721
- [ispGAL, ispGDX, and ispLSI Devices](#), on page 723
- [ispXPGA Devices](#), on page 725
- [ispXPLD5000MX and ispLSI5000VE Devices](#), on page 726
- [ispMACH and MACH Devices](#), on page 728
- [Lattice iCE40/iCE5LP Devices](#), on page 730

Fanout Guide

Lattice iCE40, ECP/EC, ispXPGA, SC/SCM, XP, MachXO, and Platform Manager Technologies

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 577](#) in the *User Guide*.

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 751](#).

GSR Resource

LatticeECP/EC, SC/SCM, XP, MachXO, Platform Manager, and ORCA Technologies

A resource for the GSR (global set reset) signal 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.

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.

Compile Point Remapping in Lattice Designs

Synplify Pro, Synplify PremierLattice iCE40, ECP/EC, SC/SCM, XP2/XP, MachXO, and Platform Manager Technologies

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 626 and [Compile Point Basics](#), on page 606 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.

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 <i>not</i> 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 <i>interface</i> 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 <i>updated</i> 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: <ul style="list-style-type: none"> • The compile point is remapped, and the Update Compile Point Timing Data option is enabled. • The interface of the compile point is changed.

Write Declared Clocks Only

By default, the Lattice mapper forward-annotates declared clocks only. Since inferred and generated clocks are not forward-annotated, they are commented out in the LPF file.

For example, suppose a design has two clocks, clk1 and clk2, where only clk1 is declared in the FDC constraint file. Since clk2 is not declared, this clock is included as a comment in the LPF file as shown below.

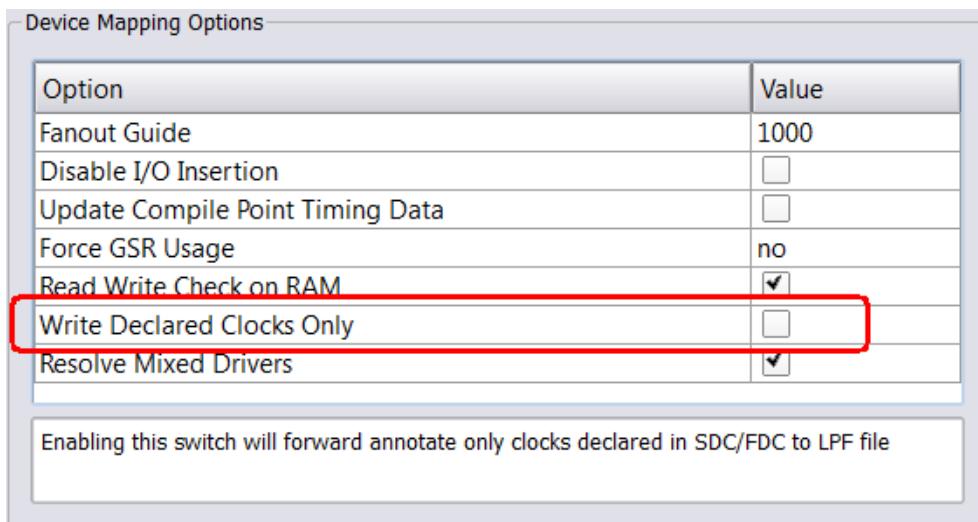
FDC File

```
create_clock {p:clk1} -period {10}
```

LPF File

```
FREQUENCY PORT "clk1" 100.0 MHz;
#FREQUENCY PORT "clk2" 200.0 MHz;
```

To override this default behavior, disable the Write Declared Clock Only option on the Device tab of the Implementation Options dialog box.



Lattice Device Mapping Options (Older Technologies)

Here are brief descriptions of the device mapping options for older Lattice technologies; for example *ispLSI* and *ispMach* families.

Map Logic to Macrocells

ispLSI1K, 2K, 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH Technologies

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.

Maximum Cell Fanin

ispLSI1K, 2K 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH Technologies.

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.

Maximum Terms/Macrocell

ispLSI1K, 2K, 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH Technologies

The Maximum Terms/Macrocell option sets the maximum number of terms per macrocell during synthesis. The default value is 16.

Percentage of Design to Optimize for Timing

ispLSI1K, 2K, 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH Technologies

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 554](#) of the *User Guide*.

ORCA Devices

This section describes guidelines for ORCA devices using the following options in the synthesis tool.

set_option Tcl Command

Lattice ORCA Technologies

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 149](#).

The `set_option` command supports the following options for ORCA devices.

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
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 711 .
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 711 .
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_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 Lattice Architectures, on page 732](#).

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_limit value
        -force_gsr yes | no | auto
        -maxfan value
        -resolve_multiple_driver 1|0
        -rw_check_on_ram 1|0
```

LatticeECP/EC and Later Devices

This section describes guidelines for using the synthesis tool with LatticeECP/EC and later technologies.

set_option Tcl Command

LatticeECP/EC and Later Technologies

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 149](#). The set_option command supports the following options for the ECP5U, ECP5UM, ECP5UM5G, LatticeECP3/ECP2S/ECP2MS/ECP2M/ECP2 and LatticeECP/EC devices.

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
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 711 .
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 711 .
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 Lattice Designs , on page 712 , for details.
Write Declared Clocks Only	Forward-annotates declared clocks only to the LPF file. By default, this option is enabled. See Write Declared Clocks Only , on page 713 , for details.

set_option Command 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 Lattice Architectures, on page 732](#).

set_option Device Options

This is the syntax for setting device options for ECP5U/ECP5UM/ECP5UM5G 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_limit value  
    -force_gsr yes | no | auto  
    -maxfan value  
    -resolve_multiple_driver 1|0  
    -rw_check_on_ram 1|0  
    -update_models_cp 1|0  
    -write_declared_clocks_only 1|0
```

LatticeSC/SCM and LatticeXP2/XP Devices

This section describes guidelines for LatticeSC/SCM and LatticeXP2/XP technologies.

set_option Tcl Command

LatticeSC/SCM, XP2/XP Technologies

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 149](#). The `set_option` command supports the following options for LatticeSC/SCM and Lattice-XP2/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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
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 711 .
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 711 .
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.
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the syn_ramstyle attribute, see syn_ramstyle , on page 509 .
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 712 , for details.
Write Declared Clocks Only	Forward-annotates declared clocks only to the LPF file. By default, this option is enabled. See Write Declared Clocks Only , on page 713 , for details.

set_option Command 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 Lattice Architectures, on page 732](#).

set_option Device Options

This is the syntax for setting device options for LatticeSC/SCM and Lattice-EXP2/XP devices.

```
set_option -technology keyword -part partName  
          -speed_grade value -package packageName  
          -block 1|0 or -disable_io_insertion 1|0  
          -fanout_limit value  
          -force_gsr yes | no | auto  
          -maxfan value  
          -resolve_multiple_driver 1|0  
          -rw_check_on_ram 1|0  
          -update_models_cp 1|0  
          -write_declared_clocks_only 1|0
```

MachXO and Platform Devices

This section describes guidelines for MachXO, MachXO2, and Platform Manager devices using the following options in the synthesis tool.

set_option Tcl Command

Lattice MachXO and Platform Manager Technologies

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 149](#).

The set_option command supports the following options for MachXO, MachXO2, and Platform Manager devices.

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
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 711 .
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 711 .
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 Lattice Designs , on page 712 , for details.
Write Declared Clocks Only	Forward-annotates declared clocks only to the LPF file. By default, this option is enabled. See Write Declared Clocks Only , on page 713 , for details.

set_option Command 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 Lattice Architectures, on page 732](#).

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_limit value
           -force_gsr yes | no | auto
           -maxfan value
           -resolve_multiple_driver 1|0
           -rw_check_on_ram 1|0
           -update_models_cp 1|0
           -write_declared_clocks_only 1|0
```

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.

set_option Command

Lattice ispGAL, ispGDX, and ispLSI Technologies

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 149](#).

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
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_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 Lattice Architectures, on page 732](#).

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
-area_delay_percent percentValue
-block 1|0 or -disable_io_insertion 1|0
-resolve_multiple_driver 1|0
-rw_check_on_ram 1|0
```

ispXPGA Devices

This section describes guidelines for using the synthesis tool with devices from the ispXPGA technology.

set_option Tcl Command

Lattice ispXPGA Technologies

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 149.

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle , on page 509.
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711.
Fanout Guide	Sets the fanout limit guideline for the open Project. The default fanout limit is 500. For information about setting fanout limits, see Lattice Attribute and Directive Summary , on page 754.
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_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 Lattice Architectures, on page 732](#).

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_limit value
-maxfan value
-resolve_multiple_driver 1|0
-rw_check_on_ram 1|0
```

LUT Function Support

Lattice ispXPGA family

For ispXPGA devices, the synthesis tool supports user-instantiated LUTs. When a hexadecimal init value is provided, it is converted to a LUT function.

ispXPLD5000MX and ispLSI5000VE Devices

This section describes guidelines for using the synthesis tool with ispXPLD5000MX and ispLSI5000VE devices.

set_option Tcl Command

Lattice ispXPLD5000MX and ispLSI5000VE Technologies

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 149](#).

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the syn_ramstyle attribute, see syn_ramstyle, on page 509 .
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
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 <code>set_option -map_logic</code> 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 <code>set_option -map_logic</code> 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 <code>set_option -map_logic</code> is set to 1. The default value is 0.
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_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 Lattice Architectures, on page 732](#).

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
           -rw_check_on_ram 1|0
```

ispMACH and MACH Devices

This section describes guidelines for using the synthesis tool with Lattice `ispMACH` and `MACH` devices.

set_option Tcl Command

Lattice ispMACH and MACH Technologies

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.

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 149](#).

Option	Description
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion, on page 711 .
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: <code>-areadelay</code> , <code>-fanin_limit</code> , <code>-maxterms</code> .
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 <code>set_option -map_logic</code> 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 <code>set_option -map_logic</code> 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 <code>set_option -map_logic</code> is set to 1. The default value is 0.
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_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 Lattice Architectures, on page 732](#).

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
           -rw_check_on_ram 1|0
```

Lattice iCE40/iCE5LP Devices

This section provides guidelines for using the synthesis tool with devices for the Lattice iCE40 and iCE5LP technology families.

- [Device Mapping Options, on page 730](#)
- [set_option Tcl Command, on page 731](#)

Device Mapping Options

Lattice iCE5LP, iCE40, iCE40LM, and iCE40UL Technologies

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 447](#) and [Constraints Panel, on page 450](#).

The following table lists device mapping options (Device tab) for the Lattice iCE40 and iCE5LP technology families.

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.
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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 707 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 577 and Controlling Buffering and Replication, on page 579 in the User Guide.
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 616 , for details.

set_option Tcl Command

Lattice iCE5LP, iCE40, iCE40LM, and iCE40UL Technologies

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. 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 Lattice Architectures, on page 732](#).

set_option Device Options

This is the syntax for setting device options for Lattice iCE40 devices.

```
set_option -technology keyword -part partName
-speed_grade value -package packageName
-block 1|0 or -disable_io_insertion 1|0
-fanout_limit value
-maxfan value
-resolve_multiple_driver 1|0
-run_prop_extract 1|0
-rw_check_on_ram 1|0
-update_models_cp 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 149](#), 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 735 for a list of valid keywords.
-part partName	Specifies a part for the implementation. Refer to Project->Implementation Options->Device or to the Lattice 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 Lattice documentation for available choices.
Optimization Options	

OPTION	DESCRIPTION
-seqshift_no_replicate 1 0	Determines whether the timing driven register replication should happen for a Sequential Shift circuit. By default, the value is set to 0 which means the register replications can happen for a sequential shift circuit.
-areadelay percent_value	Sets the percentage of nets you want optimized during synthesis for the open project. This option is available only if <code>set_option -map_logic</code> 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/O pad insertion). For additional information about disabling I/O pads, see Disable I/O Insertion , on page 711 .
-fanout_limit 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 711 .
-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 <code>auto</code> , the synthesis tool decides whether to use the global set/reset resources. The default value is <code>no</code> . For additional information about GSR, see GSR Resource , on page 711 .
-fix_gated_and_generated_clocks 1 0	Performs gated- and generated-clock optimizations when enabled. See Working with Gated Clocks, on page 828 and Optimizing Generated Clocks, on page 881 in the <i>User Guide</i> 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: <code>-areadelay</code> , <code>-fanin_limit</code> , <code>-maxterms</code> .

OPTION	DESCRIPTION
-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 <code>set_option -map_logic</code> 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 <code>set_option -map_logic</code> 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 559 in the <i>User Guide</i> 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.
-retiming 1 0	When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). Enabling <code>-retiming</code> also enables <code>-pipe</code> . See Retiming, on page 563 in the <i>User Guide</i> for a procedure.
-run_prop_extract 1 0	Determines whether the tool extracts properties for annotation. See Annotating Timing Information in the Schematic Views, on page 475 in the <i>User Guide</i> for more information.

OPTION	DESCRIPTION
-rw_check_on_ram 1 0	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
-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 712 , for details.
-write_declared_clocks_only 1 0	Forward-annotates declared clocks only to the LPF file. By default, this option is enabled. See Write Declared Clocks Only , on page 713 , for details.

Technology Keywords

The following table lists the `-technology` keyword for all Lattice technologies:

Family	Valid <code>-technology</code> Keywords
iCE40	SBTICE40
iCE40LM	SBTICE40LM
iCE5	SBTICE5LP
ispXPGA	ISPXPGA
ispGAL ispGDX ispLSI	ISPGAL, ISPGDX, ISPGDX2, PLSI1K, PLSI2K, PLSI5K, PLSI8K
ispXPLD5000MX isplsi5000VE_UPS	ISPXPLD5000MX, ISPLSI5000VE_UPS
LatticeECP/EC	ECP5U, ECP5UM, LATTICE-ECP3, LATTICE-ECP2S, LATTICE-ECP2MS, LATTICE-ECP2M, LATTICE-ECP2, LATTICE-ECP, LATTICE-EC

Family	Valid -technology Keywords
LatticeSC LatticeSCM	LATTICE-SC, LATTICE-SCM
LatticeXP	LATTICE-XP3, LATTICE-XP2, LATTICE-XP
Lattice LIFMD	LIFMD
MACH ispMACH	MACH, ISPMACH4000B, ISPMACH4000C, ISPMACH4000ZC, ISPMACH4000ZE, ISPMACH5000B, ISPMACH5000VG
MACHXO/ Platform Manager	MachXO, MachXO2, MachXO3L, MachXO3LF, Platform_Manager, Platform_Manager_2
ORCA	ORCAFpsc, ORCA2C, ORCA3C, ORCA4E

Lattice Design Options

The following options are located on the Options tab of the Implementation Options dialog box.

- [FSM Compiler](#), on page 737
- [Resource Sharing](#), on page 737
- [Pipelining](#), on page 737
- [Retiming](#), on page 738
- [Gated and Generated Clocks](#), on page 738

FSM Compiler

Turning on the FSM Compiler enables special FSM optimizations. See [Optimizing State Machines](#), on page 584 in the *User Guide*.

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 203). See [Sharing Resources](#), on page 581 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. You can pipeline multipliers and ROMs for the Lattice 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 465 for the syntax.

For detailed information about using pipelining, see [Pipelining, on page 559](#) 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 iCE40, LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager technology families.

Set global retiming either from the Project view check box or from the Device tab under the Implementation Options button. Use the `syn_allow_retimimg` attribute to enable or disable retiming for individual flip-flops. See [syn_allow_retimimg](#), on page 99 for syntax details, or [Retiming, on page 563](#) of the *User Guide*.

Pipelining is automatically enabled when retiming is enabled. Use the `syn_pipeline` attribute to control pipelining. See [syn_pipeline](#), on page 465 for syntax details.

Gated and Generated Clocks

The Clock Conversion option on the GCC tab, 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 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 828](#) 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 881](#)

Lattice Output Files and Forward Annotation

The following procedures show you how to pass information to the place-and-route tool; this information generally has no impact on synthesis. Typically, you use attributes to pass this information to the place-and-route tools. This section describes the following:

- [Synthesis Reports](#), on page 740
- [Specifying Pin Locations](#), on page 741
- [Specifying Macro and Register Placement](#), on page 742
- [Passing Technology Properties](#), on page 742
- [Specifying Padtype and Port Information](#), on page 743

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 139](#)). To view the log file, click View Log.

Specifying Pin Locations

In certain technologies you can specify pin locations that are forward annotated to the corresponding place-and-route tool. The following procedure shows you how to specify the appropriate attributes. For information about other placement properties, see [Specifying Macro and Register Placement, on page 742](#).

1. Start with a design using an appropriate Lattice technology.
2. Add the appropriate attribute to the port. For a bus, list all the bus pins, separated by commas.
 - To add the attribute from the SCOPE interface, click the Attributes tab and specify the appropriate attribute and value.
 - To add the attribute in the source files, use the appropriate attribute and syntax. For details about the attributes in the tables, see the *Attribute Reference Manual*.

Vendor Family	Attribute and Value
Lattice	loc {pin_number}

Specifying Macro and Register Placement

You can use attributes to specify macro and register placement in Lattice designs. The information here supplements the pin placement information described in [Specifying Pin Locations, on page 741](#).

For ...	Use ...
Placement of Lattice ORCA input or output registers next to I/O pads	orca_padtype define_attribute {load} orca_padtype "IBT"

Passing Technology Properties

The following table summarizes the attribute used to pass technology-specific information for certain Lattice devices.

Vendor	Attribute for passing properties
Lattice ORCA	orca_props define_attribute {p:data_in} orca_props {LEVELMODE=LVDS}

Specifying Padtype and Port Information

For certain devices, you can use the following attribute to specify technology-specific port information or padtype.

Information	Vendor	Attribute
Padtype	Lattice ORCA	orca_padtype define_attribute {AIN[3]} orca_padtype {IBT}

Integration with Lattice Tools and Flows

Use the following Lattice tools or flows for optimizing your design:

- [IP Encryption Flow for Lattice](#), on page 744
- [Inferring Lattice PIC Latches](#), on page 744
- [Controlling I/O Insertion in Lattice Designs](#), on page 751
- [Using Lattice GSR Resources](#), on page 752
- [Selective Carry-Chain Inferencing](#), on page 753

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 732. Use this procedure for details on how to incorporate Lattice encrypted IP in your design.

Inferring Lattice PIC Latches

The following procedure shows you how to control the inference of programmable I/O cells (PICs) in Lattice designs.

1. For the software to automatically infer PICs, make sure of the following:
 - The latch must be at the input port.
 - The latch must be directly driven by the input FPGA pad.
 - The design has one of the supported input control schemes: no clear or reset controls, and asynchronous clear or asynchronous resets.

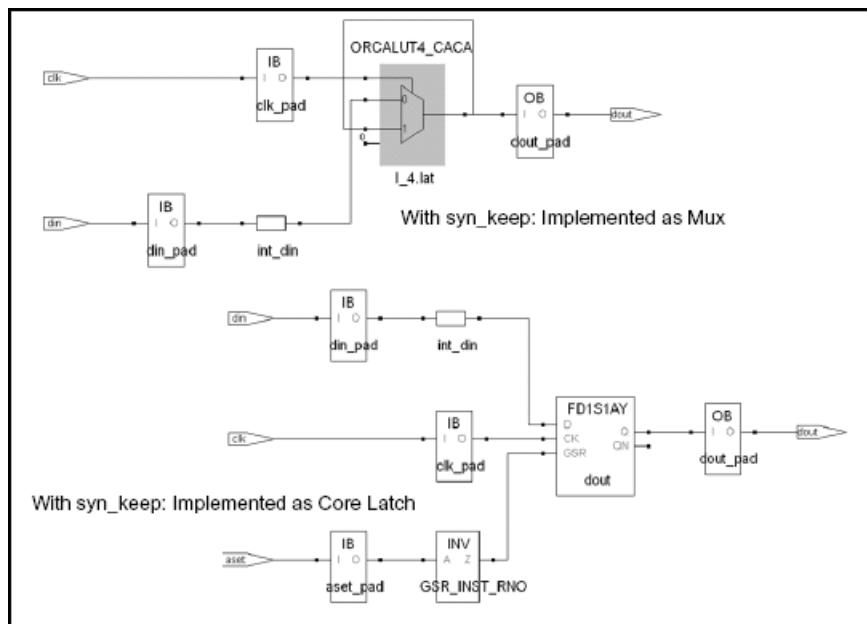
After synthesis, the tool implements the following primitives for the PICs:

- | | |
|--------|---|
| IF1S1D | Latches with asynchronous clear
Latches with GSR used for clear |
| IF1S1B | Latches with asynchronous reset
Latches with GSR used for reset. |

See [Examples of PIC Latches](#), on page 746 for examples of inferred PIC latches.

2. If you do not want to infer a PIC, set syn_keep on the input data net for the latch.

After synthesis, the tool implements the latch as either a core latch with the LATCH primitive or as a mux, depending on the Lattice technology you selected. The following figure shows an input latch with no reset or clear implemented as a mux and a core latch in different technologies.



Examples of PIC Latches

The following examples show how the tool infers PICs from the code:

- [Positive Level Data Latch with No Resets or Clears, on page 746](#)
- [Negative Level Data Latch with No Resets or Clears, on page 747](#)
- [Positive Level Data Latch with Asynchronous Reset, on page 748](#)
- [Positive Level Data Latch with Asynchronous Clear, on page 750](#)

Positive Level Data Latch with No Resets or Clears

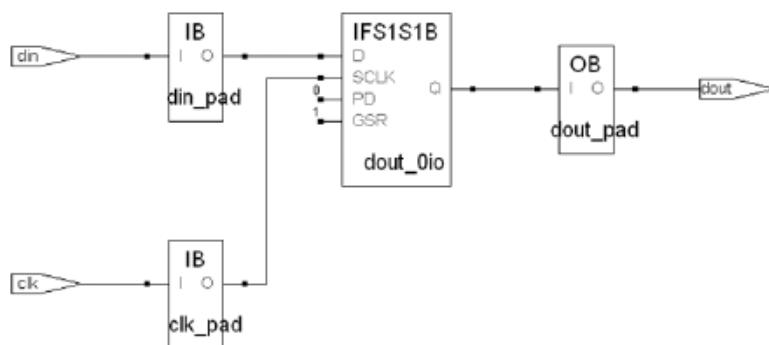
VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
  (clk : in std_logic;
  din : in std_logic;
  dout: out std_logic);
end entity inlatch;
architecture bhve of inlatch is
begin
  process(clk,din)
  begin
    if clk='1' then
      dout <= din;
    end if;
  end process;
end bhve;
```

Verilog

```
module inlatch (clk,din,dout);
  input clk;  input din;  output dout;
  reg dout;
  always @(clk)
  begin
    if(clk)
      dout <= din;
  end
endmodule
```

With this code, the tool implements the IFS1S1B latch primitive in the Technology view:



Negative Level Data Latch with No Resets or Clears

VHDL

```

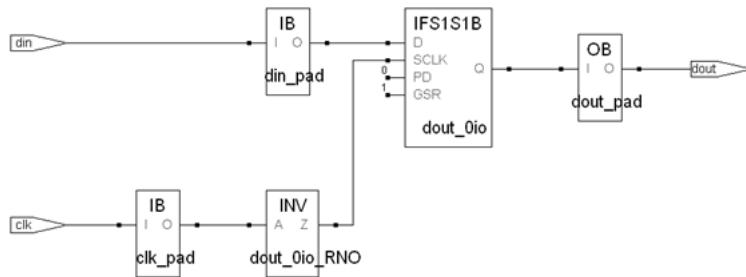
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
  (clk : in std_logic;
  din : in std_logic;
  dout: out std_logic);
end entity inlatch;
architecture bhve of inlatch is
begin
  process(clk,din)
  begin
    if clk='0' then
      dout <= din;
    end if;
  end process;
end bhve;
```

Verilog

```

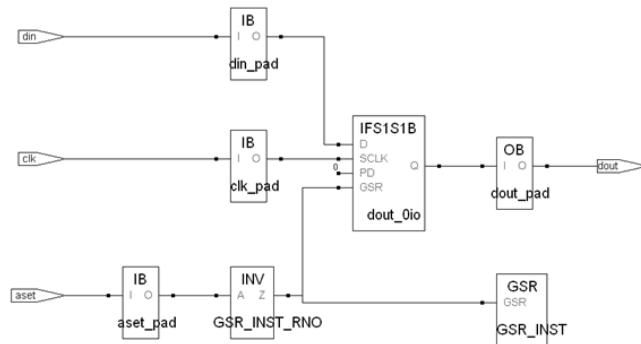
module inlatch (clk,din,dout);
  input clk;  input din;  output dout;
  reg dout;
  always @ (clk)
  begin
    if (!clk)
      dout <= din;
  end
endmodule
```

With this code, the tool implements the IFS1S1B latch primitive in the Technology view:



Positive Level Data Latch with Asynchronous Reset

The tool infers the IFS1S1B latch primitive in the Technology view with the code shown below:



VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
  (clk : in std_logic; aset: in std_logic;
  din : in std_logic; dout: out std_logic);
end entity inlatch;
architecture bhve of inlatch is
begin
  process(clk,din,aset)
  begin
    if aset ='1' then
      dout <='1';
    elsif clk='1' then
      dout <= din;
    end if;
  end process;
end bhve;
```

Verilog

```
module inlatch(clk,din,aset,dout);
  input clk; input din; input aset; output dout;
  reg dout;
  always @(clk or aset)
  begin
    if(aset)
      dout <= 1'b1;
    else if (clk)
      dout <= din;
    end
  endmodule
```

Positive Level Data Latch with Asynchronous Clear

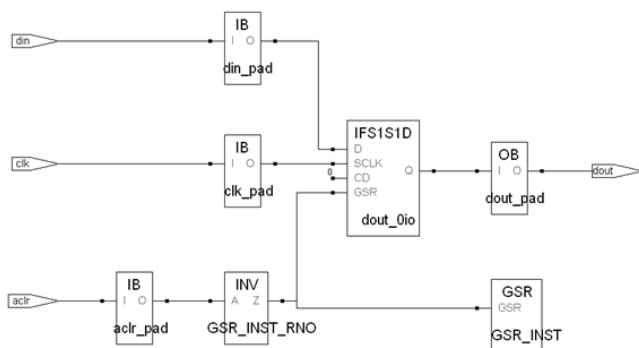
VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
(clk : in std_logic; aclr: in std_logic;
din : in std_logic; dout: out std_logic); }
end entity inlatch;
architecture bhve of inlatch is
begin
process(clk,din,aclr)
begin
if aclr ='1' then
dout <='0';
elsif clk='1' then
dout <= din;
end if;
end process;
end bhve;
```

Verilog

```
module inlatch(clk,din,aclr,dout);
input clk; input din; input aclr; output dout;
reg dout;
always @(clk or aclr)
begin
if(aclr)
dout <= 1'b0;
else if (clk)
dout <= din;
end
endmodule
```

With this code, the tool infers the IFS1S1D primitive in the Technology view:



Controlling I/O Insertion in Lattice Designs

You can control I/O insertion globally, or on a port-by-port basis.

- To control the insertion of I/O pads at the top level of the design, use the Disable I/O Insertion option as follows:
 - Select Project->Implementation Options and click the Device panel.
 - If you do not want to insert any I/O pads in the design, enable Disable I/O Insertion

Do this if you want to check the area your blocks of logic take up, before you synthesize an entire FPGA. If you disable automatic I/O insertion, you do not get *any* I/O pads in your design, unless you manually instantiate them.

 - If you want to insert I/O pads, disable the Disable I/O Insertion option.

When this option is set, the software inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist. Once inserted, you can override the I/O pad inserted by directly instantiating another I/O pad.
- To force I/O pads to be inserted for input ports that do not drive logic, follow the steps below.
 - To force I/O pad insertion at the module level, set the `syn_force_pads` attribute on the module. Set the attribute value to 1. To disable I/O pad insertion at the module level, set the `syn_force_pads` attribute for the module to 0.

- To force I/O pad insertion on an individual port, set the `syn_force_pads` attribute on the port with a value to 1. To disable I/O insertion for a port, set the attribute on the port with a value of 0.

Enable this attribute to preserve user-instantiated pads, insert pads on unconnected ports, insert bi-directional pads on bi-directional ports instead of converting them to input ports, or insert output pads on unconnected outputs.

If you do not set the `syn_force_pads` attribute, the synthesis design optimizes any unconnected I/O buffers away.

Using Lattice GSR Resources

LatticeECP/ECP2/EC, XP2/XP, SC/SCM, MachXO, and ORCA Technologies

The following procedure describes how to use GSR (global set/reset) resources and check resource usage. The GSR resource is a prerouted signal that connects to the reset input of every flip-flop, regardless of any other defined reset signals.

1. You can control the use of GSR resources as follows:
 - To improve routability and performance, use the dedicated GSR resource. Select Project ->Implementation Options and enable the Force GSR Usage option on the Device tab.
When you set this option, the synthesis software creates a GSR instance to access the resource. It uses the GSR resource for reset signals, instead of general routing. All registers are reset. when the GSR is activated, even if some flip-flops do not have a reset.
 - If a global set/reset does not correctly initialize the design, turn off the option. Select Project ->Implementation Options and disable the Force GSR Usage option on the Device tab. When this option is off, the software does not use the GSR resource unless all flip-flops have resets, and all resets use the same signal.
2. To optimize area, set the Resource Sharing option, as described in [Sharing Resources, on page 581](#).
3. To check resource usage, do the following:
 - Synthesize the design.

- Select View Log and check the Resource Usage section. For ORCA families, you can compare the LUTs in the synthesis usage report to the occupied PFUs (function units) in the report generated after placement and routing. Each PFU consists of four 4-input LUTs and four registers. An occupied PFU means that least one LUT or register was used.

Selective Carry-Chain Inferencing

Lattice ispXPLD Technologies

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, on page 697](#) for more information.

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 [Summary of Attributes and Directives, on page 19](#).

Attribute/Directive	Description
<code>black_box_pad_pin</code>	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.
<code>black_box_tri_pins</code>	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.
<code>full_case</code>	Specifies that a Verilog case statement has covered all possible cases.
<code>loc</code>	Specifies pin, register, and bus locations for Lattice I/Os.
<code>loop_limit</code>	Specifies a loop iteration limit for loops.
<code>orca_padtype</code>	Specifies the pad type for Lattice ORCA I/Os.
<code>orca_props</code>	Specifies Latticed ORCA I/O properties including CLKMODE for forward annotation.
<code>parallel_case</code>	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
<code>pragma translate_off/pragma translate_on</code>	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
<code>syn_allow_retiming</code>	Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.
<code>syn_allowed_resources</code>	Specifies the maximum number of technology-specific resources available for use in a design.
<code>syn_black_box</code>	Defines a black box for synthesis.

Attribute/Directive	Description
syn_direct_enable	<i>LatticeECP/EC, SC/SCM, XP, MachXO, ORCA, and Platform Manager families</i>
	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 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_DSPstyle	Determines how multipliers are implemented.
syn_encoding	Specifies the encoding style for state machines.
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.

Attribute/Directive	Description
<code>syn_insert_buffer</code>	Inserts a clock buffer according to the vendor-specified value.
<code>syn_insert_pad</code>	Inserts I/O buffers to either ports or nets.
<code>syn_isclock</code>	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
<code>syn_keep</code>	Prevents the internal signal from being removed during synthesis and optimization.
<code>syn_looplimit</code>	Specifies a loop iteration limit for <code>while</code> loops.
<code>syn_maxfan</code>	<p><i>LatticeECP/EC, SC/SCM, XP, MachXO, ORCA, and Platform Manager families</i></p> Sets a fanout limit for an individual input port or register output.
<code>syn_multstyle</code>	<p><i>LatticeECP, SC, P, MachXO, and Platform Manager families</i></p> Determines how multipliers are implemented.
<code>syn_netlist_hierarchy</code>	<p><i>LatticeECP/EC, SC/SCM, XP, MachXO, ORCA, and Platform Manager families</i></p> Determines if the EDIF output netlist is flat or hierarchical.
<code>syn_noarrayports</code>	<p><i>LatticeECP/EC, SC/SCM, XP, MachXO, ORCA, and Platform Manager families</i></p> Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.
<code>syn_noclockbuf</code>	Disables automatic clock buffer insertion.
<code>syn_noclockpad</code>	Infers SB_GB and SB_IO instead of SB_GB_IO on clock nets.
<code>syn_noprune</code>	Controls the automatic removal of instances that have outputs that are not driven.
<code>syn_pipeline</code>	Specifies that registers be moved into multipliers and ROMs in order to improve frequency.
<code>syn_preserve</code>	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.

Attribute/Directive	Description
<code>syn_probe</code>	Inserts probe points for testing and debugging the internal signals of a design.
<code>syn_ramstyle</code>	<p><i>LatticeECP/EC, SC/SCM, XP, MachXO, and Platform Manager families</i></p> Determines the way in which RAMs are implemented.
<code>syn_reference_clock</code>	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
<code>syn_reduce_controlset_size</code>	Controls the minimum size of the unique control-set on which control-set optimizations can occur.
<code>syn_replicate</code>	Disables replication.
<code>syn_romstyle</code>	<p><i>LatticeECP/EC, SC/SCM, XP, MachXO, Platform Manager, and iCE40 families</i></p> Determines the way in which ROMs are implemented.
<code>syn_safe_case</code>	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.
<code>syn_safefsm_pipe</code>	Removes the pipeline register on the error recovery path for the Preserve and Decode Unreachable States option.
<code>syn_sharing</code>	Specifies resource sharing of operators.
<code>syn_shift_resetphase</code>	Allows you to remove the flip-flop on the inactive clock edge, built by the reset recovery logic for an FSM when a single event upset (SEU) fault occurs.
<code>syn_srlstyle</code>	Determines how sequential shift components are implemented.
<code>syn_state_machine</code>	Determines if the FSM Compiler extracts a structure as a state machine.

Attribute/Directive	Description
<code>syn_tco<n></code>	Defines timing clock to output delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tpd<n></code>	Specifies timing propagation for combinational delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tristate</code>	Specifies that a black-box pin is a tristate pin.
<code>syn_tsu<n></code>	Specifies the timing setup delay for input pins, relative to the clock. The <i>n</i> indicates a value between 1 and 10.
<code>syn_use_carry_chain</code>	Use this attribute to turn on or off carry chain implementation for arithmetic and combinational operators.
<code>syn_useenables</code>	For CPLD technologies, generates register instances with clock enable pins for selective clock-enable inference.
<code>syn_useioff</code>	<p><i>LatticeECP/EC, SC/SCM, XP, MachXO, ORCA, and Platform Manager families</i></p> <p>Packs flip-flops in the I/O ring to improve input/output path timing.</p>
<code>syn_vote_loops</code>	<p><i>Synplify Premier</i></p> <p>Specifies whether or not to add any synchronous voter to a sequential feedback path for a distributed TMR module.</p>
<code>syn_vote_register</code>	<p><i>Synplify Premier</i></p> <p>Enables voter logic to be inserted after every register of the module specified for distributed TMR.</p>
<code>translate_off/translate_on</code>	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

APPENDIX D

Designing with Microchip

This chapter discusses the following topics for synthesizing Microchip designs:

- [Basic Support for Microchip Designs](#), on page 760
- [Microchip Components](#), on page 763
- [Microchip RAM Implementations](#), on page 772
- [Microchip Constraints and Attributes](#), on page 802
- [Microchip Device Mapping Options](#), on page 807
- [Microchip Output Files and Forward Annotation](#), on page 818
- [Integration with Microchip Tools and Flows](#), on page 824
- [Microchip Attribute and Directive Summary](#), on page 828

Basic Support for Microchip Designs

This section describes uses of the tool with Microchip devices. Topics include:

- [Microchip Device-specific Support](#), on page 760
- [Netlist Format](#), on page 760
- [Microchip Features](#), on page 762

Microchip Device-specific Support

The tool creates technology-specific netlists for a number of Microchip families of FPGAs. The following technologies are supported:

FPGAs	Technology Families
Mixed-Signal	<ul style="list-style-type: none">• SmartFusion2 and SmartFusion• Fusion
Low-Power	<ul style="list-style-type: none">• PolarFire, PolarFireSoC• IGLOO Series (IGLOO2, IGLOOE, IGLOO+, and IGLOO)• ProASIC3 Series (ProASIC3L, ProASIC3E, and ProASIC3)
Rad-Tolerant	<ul style="list-style-type: none">• RTG4• RT ProASIC3
Antifuse	<ul style="list-style-type: none">• Axcelerator• eX
Legacy	<ul style="list-style-type: none">• ACT3, ACT2/1200L, and ACT1• ProASICPLUS• 3200DX, 40MX, 42MX, 54SX, and 54SXA

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.

Netlist Format

The synthesis tool outputs EDIF or VM netlist files for use with the Microchip place-and-route application. These files have `edn` and `vm` extensions.

After synthesis the tool generates a constraint file as well, which is forward annotated as input into the Microchip place-and-route tool. These files have the following extensions:

Vendor Support	Forward Annotation Constraint File
Microchip (PolarFire)	<i>designName</i> _vm.sdc
Microchip (SmartFusion2)	<i>designName</i> _sdc.sdc and <i>designName</i> _vm.sdc
Microchip (All, except PolarFire or SmartFusion2)	<i>designName</i> _sdc.sdc

On the Implementation Results tab of the Implementation Options dialog box, two file formats: `edif` and `vm`, 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/vm
```

Targeting Output for Microchip

You can generate output targeted for Microchip.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

The following table summarizes the outputs to set for the different devices, and shows the P&R tools for which the output is intended.

Vendor Support	Output Netlist	P&R Tool
Microchip (PolarFire)	VM (.vm)	Libero SoC
Microchip (SmartFusion2)	EDIF/VM (.edn or .vm)	Libero SoC
Microchip	EDIF (.edn)	Libero SoC or IDE

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

Customizing Netlist Formats

The following table lists some attributes for customizing your Microchip output netlists:

For ...	Use ...
Netlist formatting	<p><code>syn_netlist_hierarchy</code> <code>define_global_attribute syn_netlist_hierarchy {0}</code></p>
Bus specification	<p><code>syn_noarrayports</code> <code>define_global_attribute syn_noarrayports {1}</code></p>

Microchip Forward Annotation

The synthesis tool generates Microchip-compliant constraint files from selected constraints that are forward annotated (read in and then used) by the Microchip Libero SoC or Libero IDE place-and-route software. The Microchip constraint file uses the `_vm.sdc` or `_sdc.sdc` extension. This constraint file must be imported into the Microchip flow.

By default, Microchip 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).

Microchip Features

The synthesis tool contains the following Microchip-specific features:

- Direct mapping to Microchip 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 808](#) for more information.

Microchip Components

These topics describe how the synthesis tool handles various Microchip components, and shows you how to work with or manipulate them during synthesis to get the results you need:

- [Macros and Black Boxes in Microchip Designs](#), on page 763
- [DSP Block Inference](#), on page 765
- [Control Signals Extraction for Registers \(SLE\)](#), on page 771
- [Wide MUX Inference](#), on page 771

Macros and Black Boxes in Microchip Designs

You can instantiate Smartgen¹ macros or other Microchip macros like gates, counters, flip-flops, or I/Os by using the supplied Microchip macro libraries to pre-define the Microchip macro black boxes. For certain technologies, the following macros are also supported:

- [MACC and RAM Timing Models](#)
- [SmartFusion2 MACC Block](#)
- [SmartFusion Macros](#)
- [SIMBUF Macro](#)
- [MATH18X18 Block](#)
- [Microchip Fusion Analog Blocks](#)

For general information on instantiating black boxes, see [Instantiating Black Boxes in VHDL](#), on page 399, and [Instantiating Black Boxes in Verilog](#), on page 124. For specific procedures about instantiating macros and black boxes and using Microchip black boxes, see the following sections in the *User Guide*:

- [Defining Black Boxes for Synthesis](#), on page 514
- [Using Predefined Microchip Black Boxes](#), on page 825

1. Smartgen macros now replace the ACTgen macros. ACTgen macros were available in the previous Designer 6.x place-and-route tool.

- [Using Smartgen Macros, on page 826](#)

MACC and RAM Timing Models

MACC and RAM timing models are supported for PolarFire, RTG4, SmartFusion2, and IGLOO2 devices. Timing analysis considers the timing arcs for RAM and MACC.

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 765](#).

SmartFusion Macros

The synthesis software supports the following SmartFusion macros:

- FAB_CCC
- FAB_CCC_DYN

SIMBUF Macro

ProASIC3, ProASIC3E, ProASIC3L, IGLOO, IGLOOe, IGLOO+, SmartFusion, Fusion, and Axcelerator Technologies

The synthesis software supports instantiation of the SIMBUF macro. The SIMBUF macro provides the flexibility to probe signals without using physical locations, 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.

MATH18X18 Block

The synthesis software supports instantiation of the MATH18X18 block. The MATH18X18 block is useful for mapping arithmetic functions for RTAXSDSP devices.

Microchip Fusion Analog Blocks

Microchip 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

DSP Block Inference

This feature allows the synthesis tool to infer DSP or MATH18x18 blocks for SmartFusion2 devices and MACC_PA block for PolarFire devices. The following structures are supported for SmartFusion2 devices:

- DOTP Support

The MACC block is configured in DOTP mode when two independent signed 9-bit x 9-bit multipliers are followed by addition. The sum of the dual independent 9x9 multiplier (DOTP) result is stored in the upper 35 bits of the 44-bit output. In DOTP mode, the MACC block implements the following equation:

$$P = D + (CARRYIN + C) + 512 * ((AL * BH) + (AH * BL)), \text{ when } SUB = 0$$

$$P = D + (CARRYIN + C) - 512 * ((AL * BH) + (AH * BL)), \text{ when } SUB = 1$$

Below is an example RTL which infers MACC block in DOTP mode after synthesis:

```
module dotp_add_unsign_syn (ina, inb, inc, ind, ine, dout);
parameter widtha = 6;
parameter widthb = 7;
parameter widthc = 7;
parameter widthd = 8;
parameter widthe = 30;
parameter width_out = 44;

input [widtha-1:0] ina;
input [widthb-1:0] inb;
input [widthc-1:0] inc;
input [widthd-1:0] ind;
input [widthe-1:0] ine;
output reg [width_out-1:0] dout;
always @(ina or inb or inc or ind or ine)
begin
    dout <= (ina * inb) + (inc * ind) + ine;
end
endmodule
```

The MACC block does not support DOTP mode if the

- Width of the multiplier inputs is greater than 9-bits when signed.
- Width of the multiplier inputs is greater than 8-bits when unsigned.
- Width of the non-multiplier inputs is greater than 36-bits.
- 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.

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 401](#) 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.

The following structures are supported for PolarFire devices:

- Add-mult — Adder followed by a Multiplier
- Multipliers
- Mult-adds — Multiplier followed by an Adder
- Mult-subs — Multiplier followed by a Subtractor
- Mult-acc — Multiplier followed by an Accumulator
- Wide multiplier inference
- MATH block inferencing across hierarchy
- DOTP Support
- Coefficient ROM

This section also includes the following topics:

- [Packing Coefficient ROM in the DSP, on page 768](#)
- [DSP Cascade Chain Inference, on page 768](#)

- [Multiplier-Accumulators \(MACC\) Inference, on page 769](#)

Packing Coefficient ROM in the DSP

Packing the coefficient ROM in the DSP implements the coefficient ROM data as one input to mult-add/add/sub, when inferring the MACC_PA_BC_ROM macro. The MACC_PA_BC_ROM macro extends the functionality of the MACC_PA macro to provide a 16x18 ROM at the A input. The USE_ROM pin is available for the primitive to select the input data A or the ROM data at ROM_ADDR.

Select operand A as follows:

- When USE_ROM = 0, select input data A.
- When USE_ROM = 1, select the ROM data at ROM_ADDR.

The RTL example below infers the MACC_PA_BC_ROM macro after synthesis:

```
module test(in1, rom_addr, out);
parameter data_width = 17;
parameter rom_width = 17;
parameter rom_depth = 4;

input [data_width-1:0] in1;
input [rom_depth-1:0] rom_addr;
output [47:0] out;

reg [rom_width-1:0] mem [0:2**rom_depth -1];
wire [rom_width-1:0] rom_data;

initial
begin
    $readmemb ("mem.dat", mem);
end

assign rom_data = mem [rom_addr];
assign out = rom_data * in1;
endmodule
```

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.
- 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 Microchip 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

Note: For more information about Microchip DSP math blocks along with a comprehensive set of examples, see the *Inferring Microchip RTAX-DSP MATH Blocks* application note on SolvNetPlus.

Control Signals Extraction for Registers (SLE)

The synthesis software supports extraction of control signals, the enable, synchronous set or reset, and asynchronous reset on the registers. The tool packs the enable with the EN pin, synchronous set or reset using SL_n pin and asynchronous reset using AL_n pin of the SLE.

When the fanout limit is 12, synchronous set or reset is packed using the SL_n pin. If the fanout limit is less than 12, the tool inserts extra logic for the synchronous set or reset.

The tool supports packing of the enable signal, which has higher priority than the reset signal (synchronous) of the SLE.

Initial Values for Registers (SLE)

Initial values are not supported on registers (SLE). If the initial value is specified for a register in the RTL code, the tool ignores the value and issues a warning. For the following Verilog code:

```
module test
  input clk,
  input [7:0] a,
  output [7:0] z;
  reg [7:0] z_reg = 8'hf0;
  reg one = 1'd1;
  always@ (posedge clk)
    z_reg <= a + one;
  assign z = z_reg;
endmodule
```

The initial value for register `z_reg` is specified, so the tool issues a warning message in the synthesis log report:

@W: FX1039|User-specified initial value defined for instance `z_reg[7:0]` is being ignored.

Wide MUX Inference

Wide MUXes are implemented using ARI1 primitives and is supported for PolarFire, RTG4, and SmartFusion2 technologies.

Microchip RAM Implementations

Refer to the following topics for Microchip RAM implementations:

- [RAM for PolarFire](#)
- [RAM for RTG4](#)
- [RAM for SmartFusion2/IGLOO2](#)
- [Low Power RAM Inference](#)
- [URAM Inference for Sequential Shift Registers](#)
- [Packing of Enable Signal on the Read Address Register](#)
- [Packing of INIT Value on LSRAM and URAM Blocks in PolarFire](#)
- [RAM Inference in ECC Mode](#)
- [PolarFire RAM Inference for ROM Support](#)
- [Write Byte-Enable Support for RAM](#)
- [RAM Read Enable Extraction \(ProASIC3E and Axcelerator\)](#)
- [RAM for ProASIC3/3E/3L and IGLOO+/IGLOO/IGLOOe](#)
- [RAM for ProASIC \(500K\) and ProASICPLUS \(PA\)](#)
- [RAM for Axcelerator](#)

RAM for PolarFire

The tool supports the following RAM primitives for the PolarFire device:

- RAM1K20 (LSRAM) is supported for both inference and instantiation.

The following configurations are supported for inference:

- True dual-port configuration
- Two independent data ports
- Non-ECC—1Kx20, 2Kx10, 4Kx5, 8Kx2 or 16Kx1 on each port
- Two-port configuration
- Read from port A and write to port B

- Non-ECC—512x40, 1Kx20, 2Kx10, 4Kx5, 8Kx2 or 16Kx1 on each port
 - ECC—512x33 on both ports
 - Generates SB_CORRECT and DB_DETECT flags
 - Write operations
 - Three modes—simple write, write feed-through, read before write
 - Limitations
 - Asymmetric RAM is not supported.
 - RAM64x12 (USRAM) is supported for both inference and instantiation.
- The following configurations are supported for inference:
- The RAM64x12 block contains 768 memory bits and is a two-port memory, providing one write port and one read port. Write operations for the RAM64x12 memory are synchronous. Read operations can be asynchronous or synchronous to set up the address and read out the data.
 - Consists of one read-data port and one write-data port.
 - Both read-data and write-data ports are configured to 64x12.

RAM for RTG4

The software supports the following RAM primitives for the RTG4 device:

RAM1K18_RT	Maps to RAM1K18_RT for: <ul style="list-style-type: none"> • Single-port, two-port, and dual-port synchronous read/write memory. • Read-before-write in dual-port mode for single-port and dual-port synchronous memory. • Read-enable extraction¹.
RAM64X18_RT	Maps to RAM64X18_RT for single-port, two-port, and three-port synchronous/asynchronous read and synchronous write memory.

1. Currently, read-enable extraction for wide RAM is not supported.

RAM for SmartFusion2/IGLOO2

SmartFusion2 and IGLOO2 Technologies

Two types of RAM macros are supported: 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.

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: <ul style="list-style-type: none">• 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.
Three-Port RAM Inference Support	This feature supports SmartFusion2 and IGLOO2 devices only. <ul style="list-style-type: none">• RAM64x18 is a 3-port memory that provides one Write port and two Read ports.• Write operation is synchronous, while read operations can be asynchronous or synchronous. The tool infers RAM64X18 for these structures.

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.

Three-Port RAM Inference Support

Verilog Example 1: Three-Port RAM—Synchronous Read

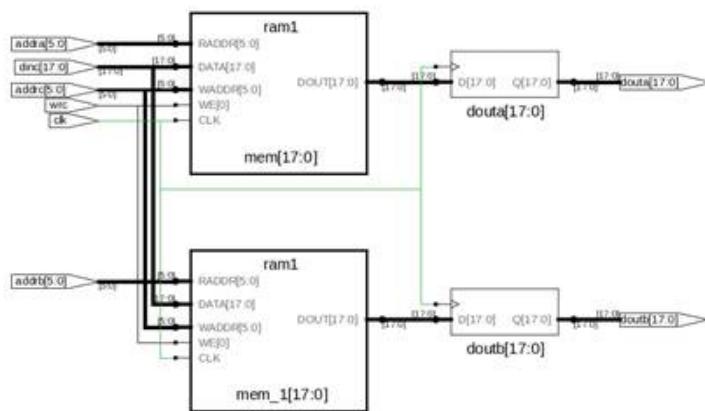
```
module ram_infer15_rtl
    (clk,dinc,douta,doutb,wrc,rda,rdb,addra,addrb,addrcc);
    input clk;
    input [17:0] dinc;
    input wrc,rda,rdb;
    input [5:0] addra,addrb,addrcc;
    output [17:0] douta,doutb;
    reg [17:0] douta,doutb;
    reg [17:0] mem [0:63];

    always@ (posedge clk)
    begin
        if(wrc)
            mem[addrcc] <= dinc;
    end

    always@ (posedge clk)
    begin
        douta <= mem[addra];
    end

    always@ (posedge clk)
    begin
        doutb <= mem[addrb] ;
    end
endmodule
```

RTL View:



The tool infers one RAM64X18.

VHDL Example 2: Three-Port RAM—Asynchronous Read

```

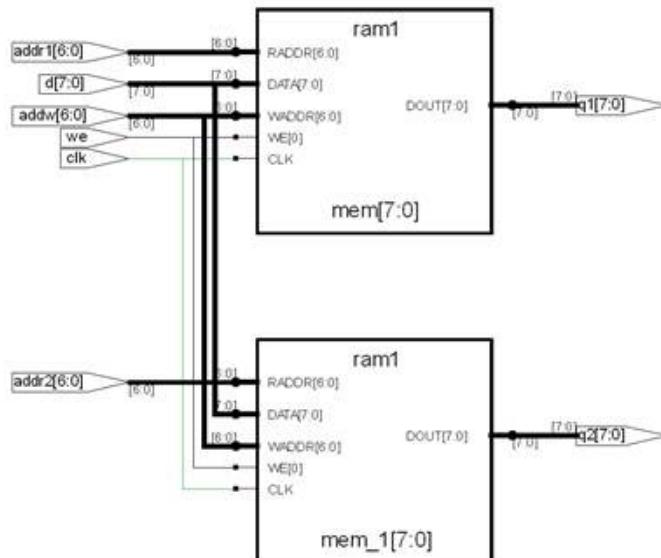
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_singleport_noreg is
port (d : in std_logic_vector(7 downto 0);
      addw : in std_logic_vector(6 downto 0);
      addr1 : in std_logic_vector(6 downto 0);
      addr2 : in std_logic_vector(6 downto 0);
      we : in std_logic;
      clk : in std_logic;
      q1 : out std_logic_vector(7 downto 0);
      q2 : out std_logic_vector(7 downto 0));
end ram_singleport_noreg;
architecture rtl of ram_singleport_noreg 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 rising_edge(clk) then
if (we = '1') then
mem(conv_integer (addw)) <= d;
end if;
end if;
end process;
end;
  
```

```

    end if;
end process;
q1<= mem(conv_integer(addr1));
q2<= mem(conv_integer(addr2));
end rtl;

```

RTL View:



The tool infers one RAM64X18.

PolarFire Asymmetric RAM support

Synthesis of asymmetric simple dual-port RAM is supported. Asymmetric RAM has different widths for read and write access ports. Read and write widths on RAM1K20 are configured independent of each other.

Two-port mode is also supported. For example, for a read configuration of 1Kx20, the following write configurations are supported:

- Write width < read width (4Kx5, 2Kx10)
- Write width > read width (512x40) (two-port mode)

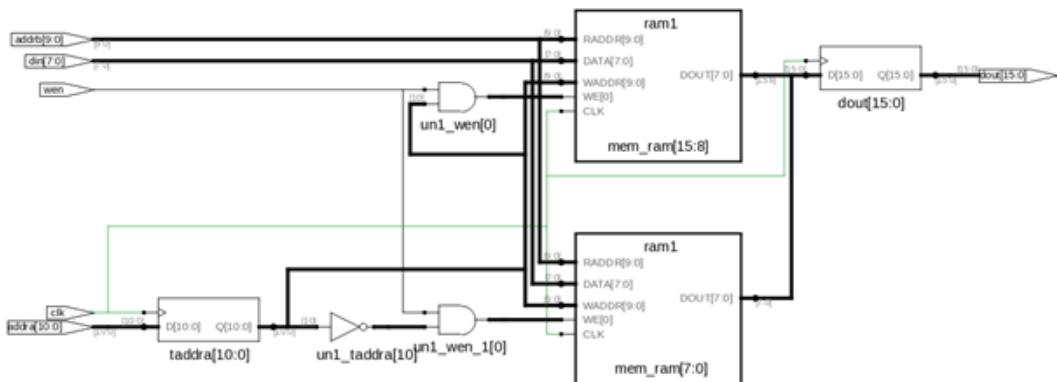
Example 1—When Write Width < Read Width

In the RTL below, write access configuration is 2Kx8 and read access configuration is 1Kx16.

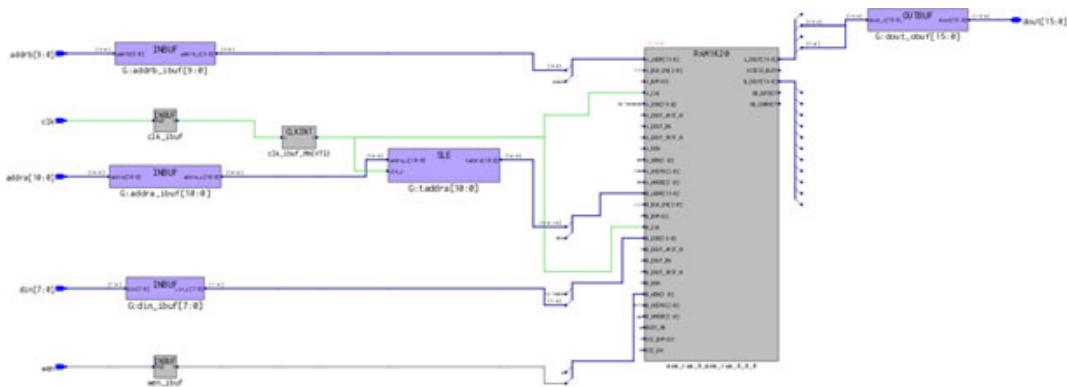
```
module asym_ram (din ,dout, addra, addrb, clk, wen);
    input [7:0] din;
    input wen;
    input [10:0] addra;
    output reg [15:0] dout;
    input [9:0] addrb;
    input clk;
    localparam ratio= 2;
    localparam max_depth=2048;
    localparam min_width=8;
    reg [10:0] taddr;
    reg [min_width-1:0] mem_ram[max_depth-1:0];
    always @ (posedge clk)
    begin
        if(wen)
            mem_ram[taddr]<=din;
        taddr<=addra;
    end

    always @ (posedge clk)
    begin // manual concatenation
        dout[min_width*0+:min_width]<=mem_ram[{0,addrb}]; // it can be
        written inside generate-loop
        dout[min_width*1+:min_width]<=mem_ram[{1,addrb}];
    end
endmodule
```

RTL View



Technology View

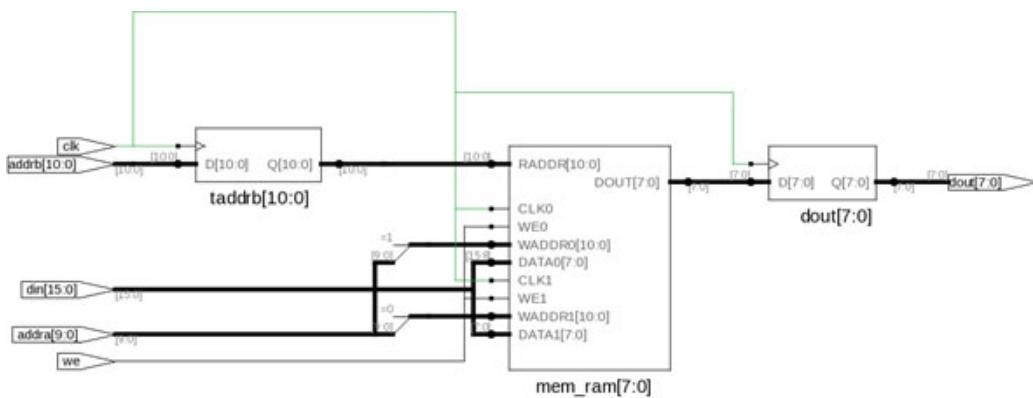


Example 2—When Write Width > Read Width

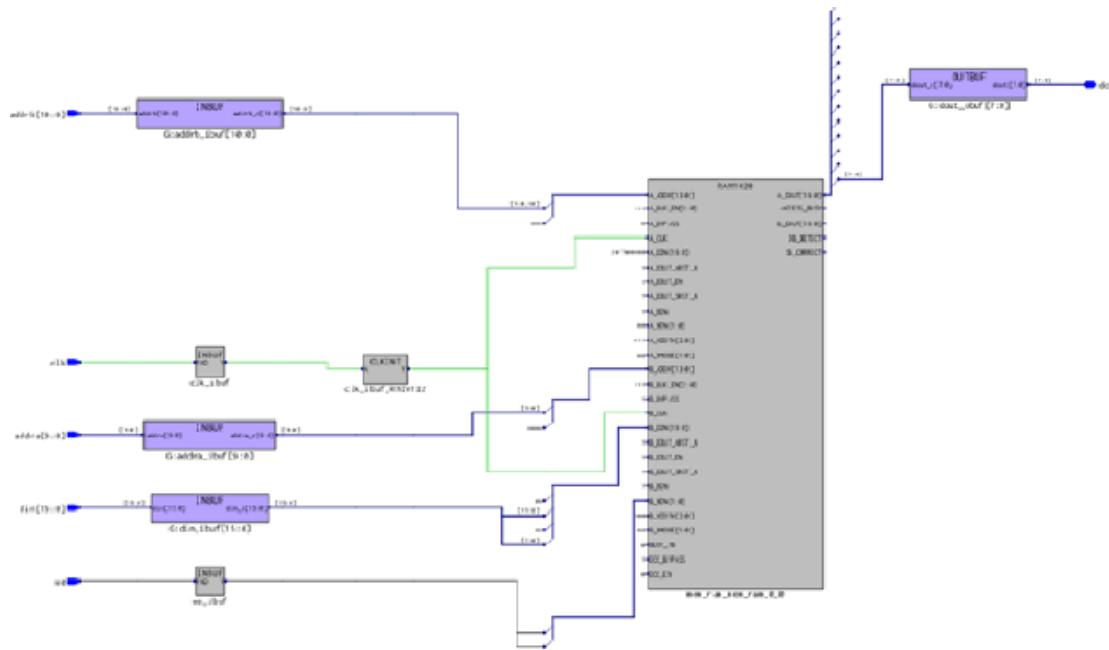
The code below implements asymmetric RAM with 1Kx16 write access and 2Kx8 read access configurations.

```
module asym_ram(din, dout, addra, addrb, clk, we);
    input [15:0] din;
    input [9:0] addra;
    output reg [7:0] dout;
    input [10:0] addrb;
    input clk;
    input we;
    localparam max_depth=2048;
    localparam min_width=8;
    reg [min_width-1:0] mem_ram[max_depth-1:0];
    reg [9:0] taddr;
    reg [10:0] taddrb;
    always @(posedge clk)
    begin
        dout<=mem_ram[taddrb];
        taddrb<=addrb;
    end
    always @(posedge clk)
    if (we)
    begin
        mem_ram[{0,addra}]<=din[min_width*0+:min_width];
        mem_ram[{1,addra}]<=din[min_width*1+:min_width];
    end
endmodule
```

RTL View



Technology View



Attributes

RAM attributes, like `syn_ramstyle`, are applied to control the inference.

Read Write Control Signals

Control signals are the same as that of symmetric RAM implementation:

- Enable read and write
- Synchronous and asynchronous reset on RAM registers
- RAM read-write mode (no change, write-first, read-first)
- Packing of RAM registers or pipelines

Limitations

- Initial value is not supported.
- Asymmetric true dual-port RAM is not supported.

- Read/write logic check creation is not supported. If the read/write check option is enabled, then the RAM is implemented in symmetric mode.

Low Power RAM Inference

PolarFire, RTG4, SmartFusion2, and IGLOO2 Technologies

Enhanced RAM inference uses the BLK pin of the RAM for reducing power consumption. By setting the global option `low_power_ram_decomp 1` in the project file, the tool fractures the wide RAMs on the address width, using the BLK pin of the RAM to reduce power consumption. By default, the tool fractures wide RAMs by splitting the data width to improve timing.

This feature is supported for single-port, simple-dual port, and true-dual port RAM modes.

URAM Inference for Sequential Shift Registers

PolarFire Technologies

URAM inference is supported for sequential shift registers.

By default, seqshift is implemented using registers. The `syn_srlstyle` attribute is used to override the default behavior of seqshift implementation using URAM. This attribute can be applied on the top-level module or on a seqshift instance in the RTL view, by dragging and dropping the instance to the SCOPE editor.

If the attribute is applied on the top-level module, the tool infers URAM for all the seqshifts in the design using the following threshold values:

Depth ≥ 8 and Depth*Width > 84

If the attribute is applied on the seqshift instance, the tool infers URAM irrespective of the threshold values.

syn_srlstyle Values

Technology	Value	Description
PolarFire	URAM	• Infers seqShift register components as RAM64X12.

syn_srlstyle Syntax

FDC	define_attribute {object} syn_srlstyle {registers uram } define_global_attribute syn_srlstyle {registers uram }
Verilog	object /* synthesis syn_srlstyle = "registers uram " */;
VHDL	attribute syn_srlstyle : string; attribute syn_srlstyle of object : signal is "registers uram ";

Example

The tool infers a seqshift primitive for the following HDL:

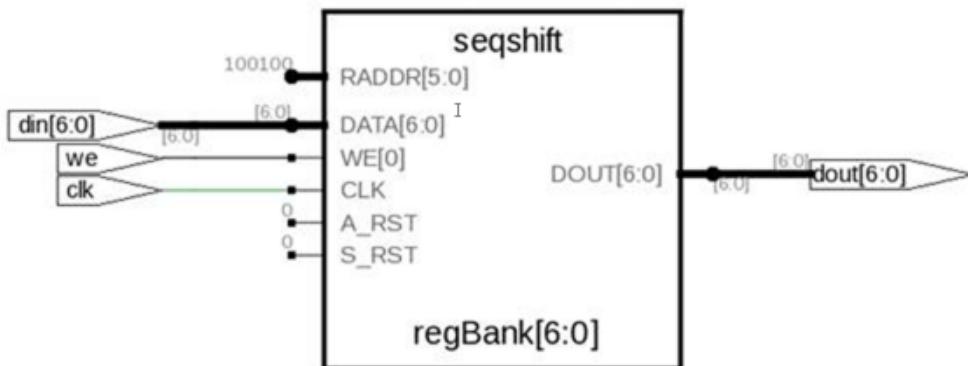
```
module p_seqshift(clk, we, din, dout);
parameter SRL_WIDTH = 7;
parameter SRL_DEPTH = 37;

input clk, we;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0]
    /*synthesis syn_srlstyle = "uram"*/;
integer i;

always @ (posedge clk) begin
    if (we) begin
        for (i=SRL_DEPTH-1; i>0; i=i-1) begin
            regBank[i] <= regBank[i-1];
        end
        regBank[0] <= din;
    end
end

assign dout = regBank[SRL_DEPTH-1];
endmodule
```

The seqshift generated for the HDL above is shown in technology view.



Limitations

Limitations include the following:

- URAM inference for seqshift is not supported, if the output comes from a dynamic stage.
- Seqshifts with synchronous reset or asynchronous reset are inferred as registers.
- Seqshifts with both synchronous reset and asynchronous reset are inferred as registers.
- Seqshifts with both reset and set are inferred as registers.
- Seqshifts with enable signal having higher priority than synchronous set or synchronous reset are inferred as registers.

Packing of Enable Signal on the Read Address Register

PolarFire and RTG4 Technologies

The tool packs the enable signal on the read address register for the following:

- [PolarFire RAM1K20 and RAM64x12 Enhancements](#)
- [RTG4 RAM64x18, RAM64x18_RT, RAM1K18_RT Enhancements](#)

PolarFire RAM1K20 and RAM64x12 Enhancements

The tool supports the packing of enable signal on the read address register into RAM1K20 (A_REN) and RAM64x12 (R_ADDR_EN).

RTG4 RAM64x18, RAM64x18_RT, RAM1K18_RT Enhancements

Packing of enable signal on the read address register into RAM1K18_RT (A_REN), RAM64x18 (A_ADDR_EN & B_ADDR_EN), and RAM64x18_RT (A_ADDR_EN & B_ADDR_EN) is supported.

Packing of INIT Value on LSRAM and URAM Blocks in PolarFire

INIT value packing is supported for RAM1K20 and RAM64x12 RAM blocks in the PolarFire device. It is also supported for ECC mode of RAM1K20 and SeqShifts inferred as URAMs. Here is some sample code:

```
module test (clk,we,waddr,raddr,din,q);
  input clk,we;
  input [addr_width - 1 : 0] waddr,raddr;
  input [data_width - 1 : 0] din;
  output [data_width - 1 : 0] q;
  reg [data_width - 1 : 0] q;
  reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0];
  initial $readmemb("mem1.dat", mem);
  always @ (posedge clk)
    if(we) mem[waddr] <= din;
  always @ (posedge clk )
    if(we) q <= din;
    else q <= mem[raddr];
endmodule
```

RAM Inference in ECC Mode

PolarFire and RTG4 Technologies

Enable Error Correction Code (ECC) using the `syn_ramstyle` attribute with the value shown below.

Attribute	Value	Description	Scope
<code>syn_ramstyle</code>	ECC	Enables RAM inference with ECC	FDC: Global, view, instance HDL: Module, architecture, memory

ECC Error Flag Generation

To implement ECC error flag generation, use the following commands:

Command	Argument	Description
<code>syn_create_err_net</code>	<code>-name newNetName</code>	Specifies the new net name to which the generated error flag is connected.
	<code>-inst i:RAMInstance</code>	Specifies the instance name of the high reliability module for error monitoring.
	<code>[-single_bit -double_bit]</code>	Specifies monitoring of a single-bit or double-bit error flag or both.
<code>syn_connect</code>	<code>-from n:newNetName</code>	Specifies the net name created by the <code>syn_create_err_net</code> command.
	<code>-to [n:existingNet t:submodulePort p:topPort]</code>	Specifies the destination for the generated error flag, which can be: <ul style="list-style-type: none"> • An existing net or a submodule • Output port or a top-level output port

To control single-bit and double-bit error flag generation, use the `-single_bit/-double_bit` argument of the `syn_create_err_net` command.

-inst	-single_bit double_bit	Description
i:ECCRAMInstance	None Both	Create one error flag by ORing the following ports of all the SRAM blocks: A_SB_CORRECT B_SB_CORRECT A_DB_DETECT B_DB_DETECT
	-single_bit	Create one error flag by ORing the following ports of all the SRAM blocks: A_SB_CORRECT B_SB_CORRECT
	-double_bit	Create one error flag by ORing the following ports of the SRAM blocks: A_DB_DETECT B_DB_DETECT

Example 1: Using ECC with FDC Constraints

```

module test (clka,clkb,rst,wea,addra,dataina,qa,web,
            addrb,datainb,qb,error_flag);
parameter addr_width =10;
parameter data_width = 16;
input clka,clkb,wea,web,rst;
input [data_width - 1:0] dataina,datainb;
input [addr_width - 1:0] addra,addrb;
output error_flag;
output reg [data_width - 1:0] qa,qb;
reg [data_width - 1 : 0] mem [(2**addr_width) - 1:0]
    /* synthesis syn_ramstyle = "ecc" */;
always @ (posedge clka)
begin
    if(wea) mem[addra] <= dataina;
end
always @ (posedge clkb)
begin
    if (~rst)
        qa <=16'd0;

```

```

else begin
    if (~wea)
        qa <= mem[addrb];
    end
end
endmodule

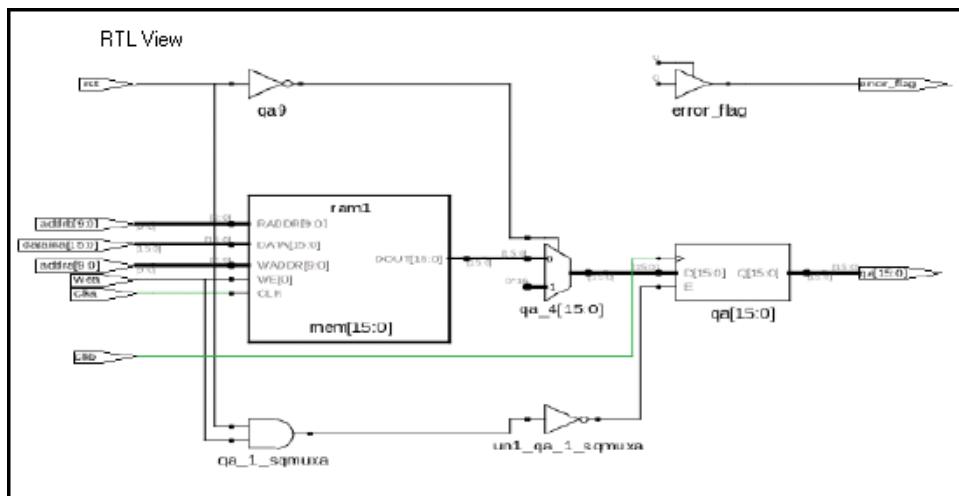
```

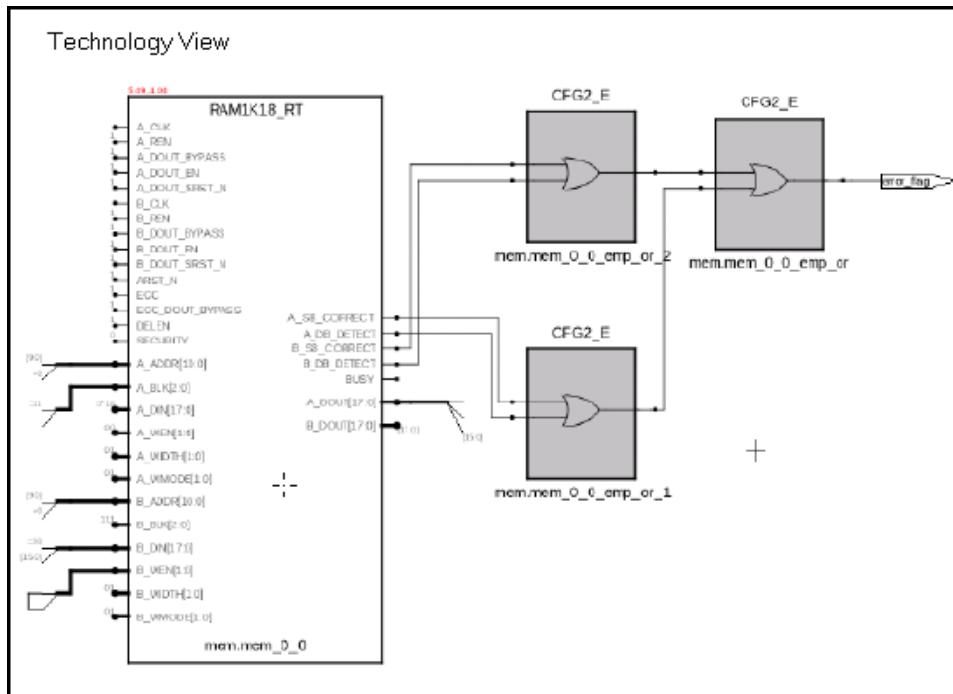
For the code above, specify the following constraints in the FDC file, where one error flag is specified for both single-bit and double-bit errors:

```

define_global_attribute {syn_ramstyle} {ecc}
syn_create_err_net {-name {error_net} -inst {i:mem[15:0]}}
syn_connect {-from {n:error_net} -to {p:error_flag}}

```

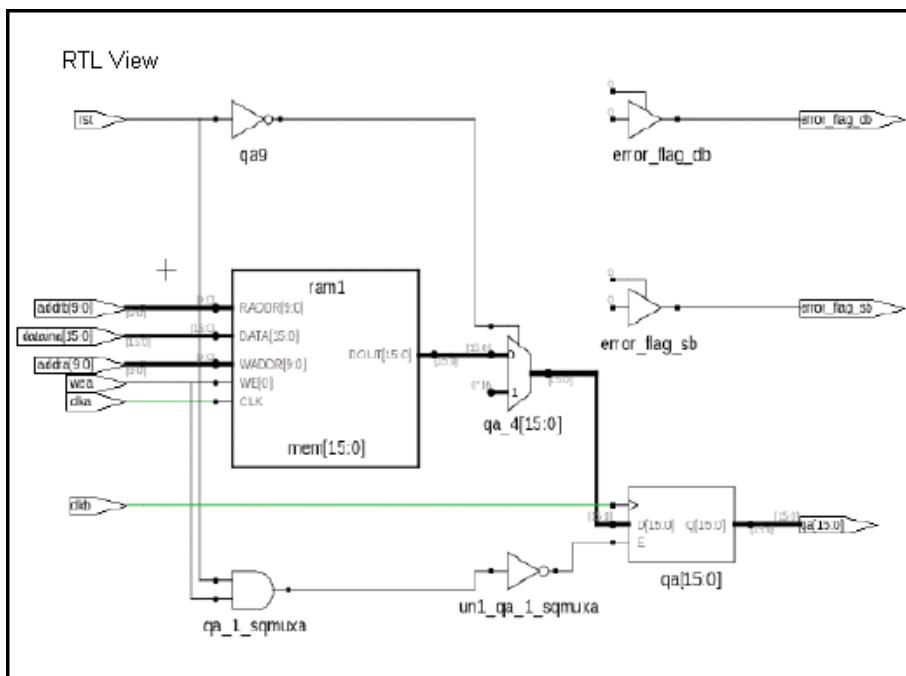


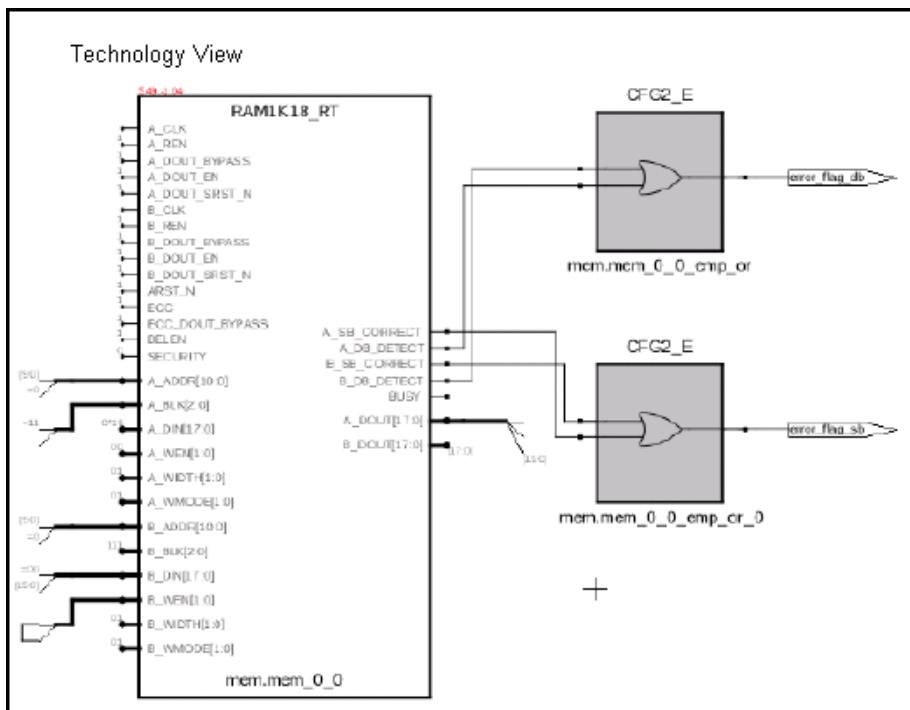


Example 2: ECC with Single- and Double-bit Errors

For the code above, specify separate error flags for single-bit and double-bit errors.

```
define_attribute {i:mem[15:0]} {syn_ramstyle} {ecc}
syn_create_err_net {-name {error_net_sb} -inst {i:mem[15:0]} -single_bit}
syn_connect {-from {n:error_net_sb} -to {p:error_flag_sb}}
syn_create_err_net {-name {db_error_net_db} -inst {i:mem[15:0]} -double_bit}
syn_connect {-from {n:db_error_net_db} -to {p:db_error_flag_db}}
```





Along with ECC, you can enable the Single Event Transient (SET) mitigation using the `syn_ramstyle` attribute.

Attribute	Value	Description	Scope
<code>syn_ramstyle</code>	ECC, SET	Enables RAM inference with ECC or SET mitigation	FDC: Global, view, instance HDL: Module, architecture, memory

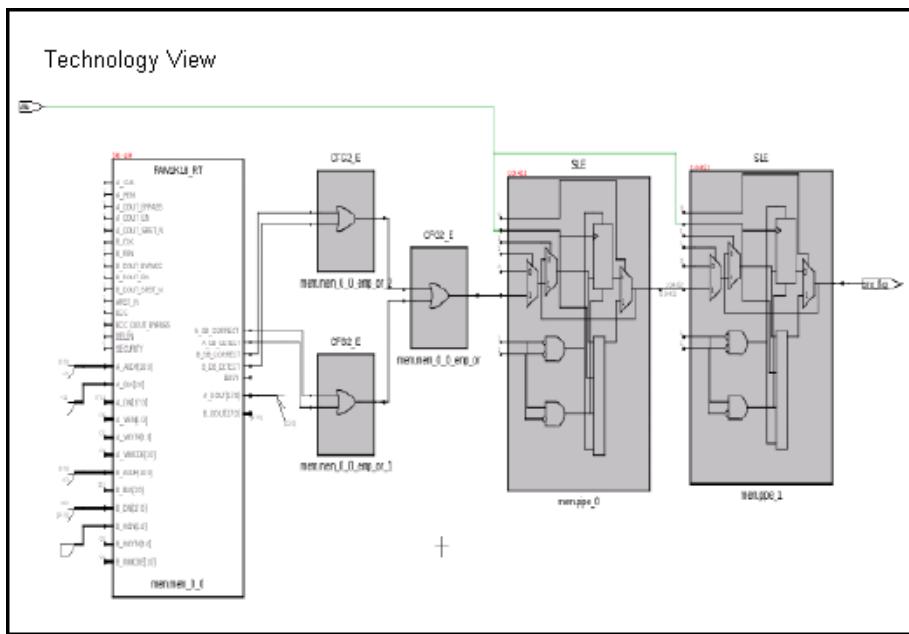
Pipeline stages can be inserted on the error flag path by including the bold options below with the `syn_create_err_net` command.

Command	Argument	Description
syn_create_err_net	-name <i>newNetName</i>	Specifies the new net name to which the generated error flag is connected.
	-inst <i>i:RAMInstance</i>	Specifies the instance name of the high reliability module for error monitoring.
	[-single_bit -double_bit]	Specifies monitoring of a single-bit or double-bit error flag or both.
	[-err_pipe_num {#}]	{#} Specifies the number of pipeline stages to be introduced in the error flag path.
	[-err_clk {n:clockToThePipelineRegisters}]	Specifies the clock signal for the pipeline registers.

Example 3: Pipeline Stages in the Error Flag Path with FDC Constraints

For the code above, specify the following FDC constraints to insert two pipeline stages in error flag path:

```
define_global_attribute {syn_ramstyle} {ecc}
syn_create_err_net {-name {error_net} -inst {i:mem[15:0] -err_pipe_num {2}
-err_clk {n:clkb}}}
syn_connect {-from {n:error_net} -to {p:error_flag}}
```



Limitations

Inferring ECC pipeline stages (ECC_DOUT_BYPASS = 0) is not supported.

PolarFire RAM Inference for ROM Support

By default, ROM is implemented using RAM1K20 and RAM64x12 depending on the RAM threshold values. The RAM is inferred in non-low (speed) mode. Asynchronous ROM is always mapped to RAM64x12.

Use the `syn_romstyle` attribute to override the default behavior of the ROM implementation with RAM or logic.

The `syn_romstyle` attribute can be used to determine the implementation of the ROM components as follows:

FDC	define_attribute {object} syn_romstyle {logic uram lsram} define_global_attribute syn_romstyle {logic uram lsram}
Verilog	object /* synthesis syn_romstyle = "logic uram lsram" */ ;
VHDL	attribute syn_romstyle : string; attribute syn_romstyle of object : signal is "logic uram lsram";

The `syn_romstyle` values are:

Value	Description
logic	ROM is inferred as registers or LUTs.
uram lsram	ROM is inferred as RAM1K20 or RAM64x12. Asynchronous ROM is mapped to RAM64x12 even if the lsram attribute is applied.

Example 1

```
module test(clk,addr,dataout);
  input clk;
  parameter addr_width = 10;
  parameter data_width = 20;
  input [addr_width-1:0] addr;
  output [data_width-1:0] dataout;
  reg [data_width-1:0] dataout;
  always @ (posedge clk)
    case (addr)
      10'd0 : dataout <= 20'b01000110000010001100;
      10'd1 : dataout <= 20'b11100000110110011100;
      10'd2 : dataout <= 20'b10110101101111011001;
      10'd3 : dataout <= 20'b01111010011000000000;
      10'd4 : dataout <= 20'b00110110100111111100;
      10'd5 : dataout <= 20'b11110101000010001010;
      10'd6 : dataout <= 20'b00010010110101000110;
      10'd7 : dataout <= 20'b01001001010010100110;
      10'd8 : dataout <= 20'b01110111000111111011;
      10'd9 : dataout <= 20'b1001010111110111110;
      ...
      ...
      10'd1015 : dataout <= 20'b1101101000011111101;
      10'd1016 : dataout <= 20'b11001000101001110111;
      10'd1017 : dataout <= 20'b01010000111100100011;
      10'd1018 : dataout <= 20'b11000110011011011011;
      10'd1019 : dataout <= 20'b10000000110101100110;
```

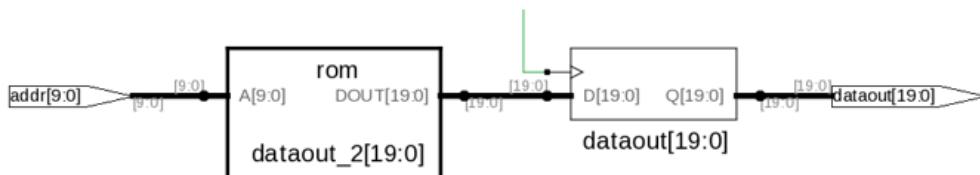
```

10'd1020 : dataout <= 20'b11100101010001001011;
10'd1021 : dataout <= 20'b10010011000110001010;
10'd1022 : dataout <= 20'b00100000110010000101;
10'd1023 : dataout <= 20'b10001010000011111010;
default : dataout <= 20'b000000000000000000000000;
endcase

endmodule

```

The following ROM is displayed in the SRS view of the tool for the RTL above. The tool infers RAM1K20 for the ROM below.



Example 2

```

module test (addr,dataout);
parameter addr_width = 8;
parameter data_width = 10;
input [addr_width - 1 : 0] addr;
output [data_width - 1 : 0] dataout;
reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0];
initial $readmemh("mem256x10_hex.list", mem);
assign dataout = mem[addr];
endmodule

```

The following ROM is displayed in the SRS view of the tool for the RTL above. Since this is an asynchronous ROM, the tool infers RAM64x12.



Write Byte-Enable Support for RAM

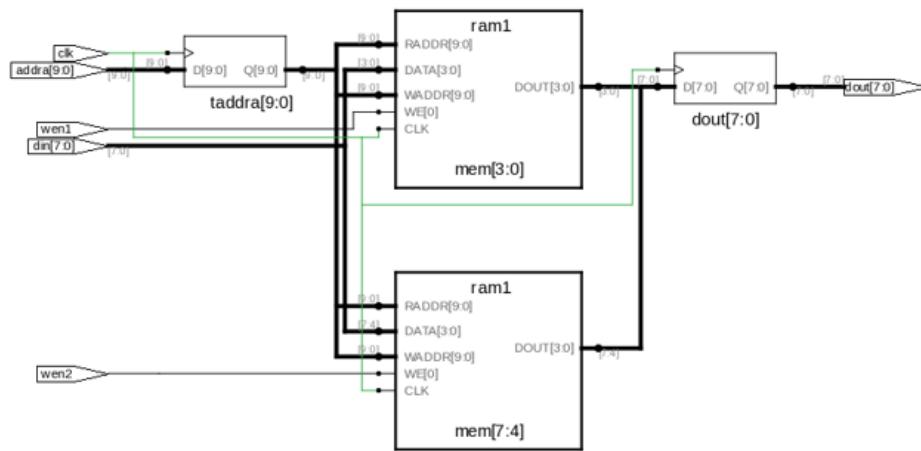
For RAM with n write enables used to control writing of data into memory locations, the compiler creates n sub-instances of the RAM with different write enables. The mapper merges these multiple RAM blocks into single or multiple block RAM, depending on the threshold and number of write enables. The write byte-enable pin (A_WEN/B_WEN [1:0]) of the block RAM primitives are configured to control the write operation for block RAMs.

Example

```
module ram (din, dout, addra, addrb, clk, wen1, wen2);
    input [7:0] din;
    input wen1;
    input wen2;
    input [9:0] addra;
    input clk;
    output reg [7:0] dout;
    localparam max_depth=1024;
    localparam min_width=8;
    reg [9:0] taddra;
    reg [min_width-1:0] mem_ram[max_depth-1:0];

    always @ (posedge clk)
    begin
        taddra<=addra;
        if(wen1)
            mem_ram[taddra] [3:0]<=din[3:0];
        if(wen2)
            mem_ram[taddra] [7:4]<=din[7:4];
    end
    always @ (posedge clk)
    begin
        dout <= mem_ram[taddra];
    end
endmodule
```

The compiler infers two ram1 shown in the SRS view below, which can be combined and mapped into a single RAM1K18_RT or RAM1K20.



RAM Read Enable Extraction (ProASIC3E and Axcelerator)

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

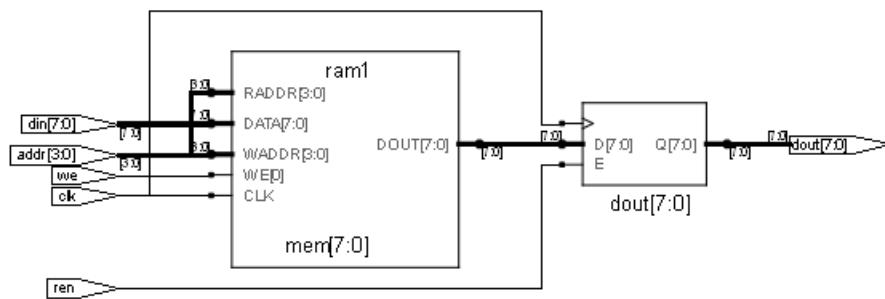
```

```

if (we)
    mem[addr] <= din;
    if (ren)
        dout = mem[addr];
end

endmodule

```



RAM for 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
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);

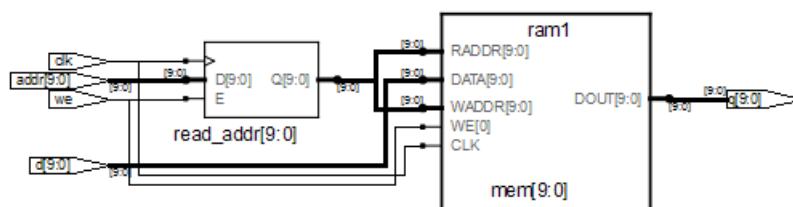
begin

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:

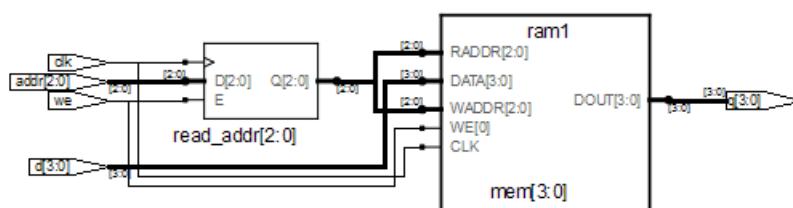
```

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;

```



RAM for ProASIC (500K) and ProASICPLUS (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

RAM for 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.

Microchip Constraints and Attributes

The synthesis tools let you specify timing constraints, general HDL attributes, and Microchip-specific attributes to improve your design. You can manage the attributes and constraints in the SCOPE interface. The following topics explain how to implement constraints and attributes for Microchip designs. Refer to:

- [Microchip I/O Standards](#), on page 802
- [Global Buffer Promotion](#), on page 803
- [The syn_maxfan Attribute in Microchip Designs](#), on page 804
- [Radiation-tolerant Applications](#), on page 805

Microchip I/O Standards

Fusion, IGLOO family, ProASIC3 family, ProASIC, ProASICPLUS, and Axcelerator Families

Microchip has vendor-specific I/O standard constraints it supports for synthesis. 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.

ProASIC3L, ProASIC3E, IGLOOe, Fusion	IGLOO, IGLOO+, ProASIC3	ProASIC, ProASICPLUS	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

ProASIC3L, ProASIC3E, IGLOOe, Fusion	IGLOO, IGLOO+, ProASIC3	ProASIC, ProASIC^{PLUS}	Axcelerator
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 321](#) for a list of industry I/O standards.
- [Microchip Attribute and Directive Summary, on page 828](#) for a list of Microchip attributes and directives.

Global Buffer Promotion

PolarFire, RTG4, SmartFusion2, IGLOO2 Technologies

The synthesis software inserts the global buffer (CLKINT) on clock, asynchronous set/reset, and data nets based on a threshold value. The supported 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.

Net	Global buffer inserted for threshold value > or =
PolarFire Devices	
Clock	2
Asynchronous Set/Reset	6
Data	5000
RTG4 Devices	
Clock	2
Asynchronous Set/Reset	200000
Data	5000
SmartFusion2 and IGLOO2 Devices	
Clock	2
Asynchronous Set/Reset	12
Data	5000

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="CLKINT"` so that the tool inserts a global buffer on the particular net, which is less than the threshold value. You can only specify CLKINT as a valid value for SmartFusion2 devices.

The `syn_maxfan` Attribute in Microchip 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.

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 Microchip 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

For details, see:

- [Working with Microchip Radhard Designs, on page 927](#)
- [syn_radhardlevel, on page 495](#)

Microchip Device Mapping Options

To achieve optimal design results, set the correct implementation options. Some options include the following:

- [Promote Global Buffer Threshold](#), on page 807
- [I/O Insertion](#), on page 808
- [Update Compile Point Timing Data Option](#), on page 809
- [Operating Condition Device Option](#), on page 811

See Also

- [Microchip set_option Command Options](#), on page 813
- [Microchip Tcl set_option Command Options](#), on page 814

Promote Global Buffer Threshold

SmartFusion, Fusion, IGLOO, and ProASIC3 Technologies

The Promote Global Buffer Threshold option is used 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.

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, IGLOO2, and RTG4 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, IGLOO2, and RTG4 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 and IGLOO2 devices are the following:

Net	Global buffer inserted for threshold value > or =
Clock	2
Asynchronous Set/Reset	12
Data	5000

The threshold values for RTG4 devices are the following:

Net	Global buffer inserted for threshold value > or =
Clock	2
Asynchronous Set/Reset	200000
Data	5000

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="CLKINT"` so that the tool inserts a global buffer on the particular net, which is less than the threshold value. You can specify CLKINT, RCLKINT, CLKBUF, or CLKIBUF as values for SmartFusion2, RTG4, and IGLOO2 devices.

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 Microchip I/O pads you need directly. If you manually insert I/O pads, you only insert them for the pins that require them.

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, and ProASIC3 technology families.

You enable/disable global retiming with the Retiming device mapping option (Project view or Device panel). You can use the `syn_allow_retimimg` attribute to enable or disable retiming for individual flip-flops. See [syn_allow_retimimg, on page 99](#) and the *User Guide* for more information.

Update Compile Point Timing Data Option

PolarFire, SmartFusion2, SmartFusion, Fusion, IGLOO2, IGLOOE, IGLOO+, IGLOO, ProASIC3, ProASIC (500K), and ProASICPLUS (PA) Technologies

The Update Compile Point Timing Data option used with 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 626](#) 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.

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.

Automatic Compile Points

PolarFire Technology

This feature is enabled, by default, only for PolarFire devices.

The tool supports the Automatic Compile Points (ACP) flow. For details, see [The Automatic Compile Point Flow, on page 627](#) in the *User Guide*.

Operating Condition Device Option

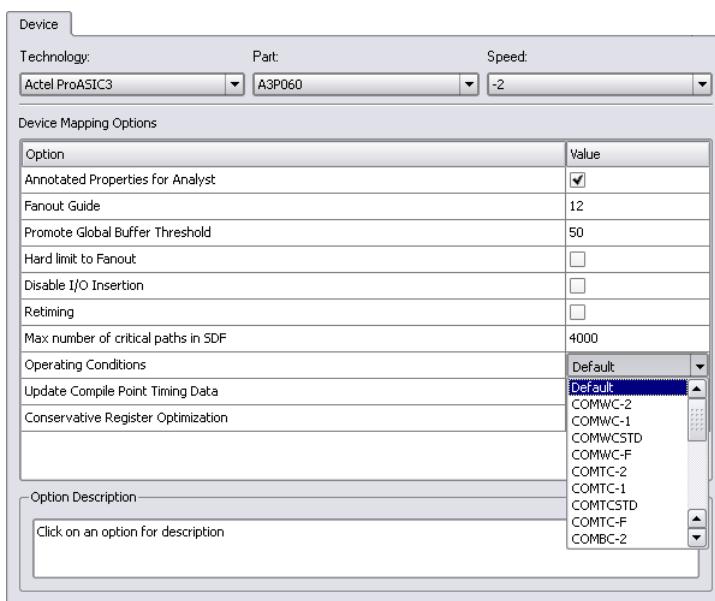
You can specify an operating condition for certain Microchip technologies:

- PolarFire
- RTG4
- SmartFusion2, SmartFusion, FUSION
- IGLOO2/IGLOO+/IGLOO/IGLOOe
- ProASIC^{PLUS} (PA), ProASIC3/3E/3L
- ProASIC (500K)
- 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.



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 Microchip 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.

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 Microchip's documentation or software for information on valid combinations and more information on the meaning of each operating condition.

Microchip set_option Command Options

To select device mapping options for Microchip technologies, select Project -> Implementation Options->Device and set the options.

Option	For details, see ...
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the syn_ramstyle attribute, see syn_ramstyle, on page 509 .
Conservative Register Optimization	See the Microchip Tcl set_option Command Options , on page 814 for more information about the preserve_registers option.
Disable I/O Insertion	I/O Insertion , on page 808 .

Option	For details, see ...
Fanout Guide	Setting Fanout Limits, on page 577 of the <i>User Guide</i> and The syn_maxfan Attribute in Microchip Designs , on page 804.
Operating Conditions (certain technologies)	Operating Condition Device Option , on page 811
Promote Global Buffer Threshold	Controlling Buffering and Replication, on page 579 of the <i>User Guide</i> and Promote Global Buffer Threshold , on page 807.
Retiming	Retiming, on page 563 of the <i>User Guide</i>
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	Synplify Pro, Synplify Premier Update Compile Point Timing Data Option , on page 809

Microchip 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 [Implementation Options Command, on page 444](#)).

This section describes the Microchip-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 Microchip architectures. For a complete list of options for this command, refer to [set_option, on page 149](#). You cannot specify a package (-package option) for some Microchip technologies in the synthesis tool environment. You must use the Microchip back-end tool for this.

Option	Description
-technology keyword	Sets the target technology for the implementation. Keyword must be one of the following Microchip architecture names: 3200DX, 40MX, 42MX, 500K, 54SX, 54SXA, ACT1, ACT2, ACT3, AXCELERATOR, EX, FUSION, IGLOO, IGLOOE, IGLOO+, IGLOO2, PA, ProASIC3, ProASIC3E, ProASIC3L, SmartFusion, SmartFusion2, RTG4, PolarFire. 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	<i>Axcelerator, Fusion, IGLOO, ProASIC3, SmartFusion, RTG4, and PolarFire families</i> Specifies the package. Refer to Project-> Implementation Options->Device for available choices.
-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.
-disable_io_insertion 1 0	Prevents (1) or allows (0) insertion of I/O pads during synthesis. The default value is false (enable I/O pad insertion). For additional information about disabling I/O pads, see I/O Insertion , on page 808 .
-fanout_limit value	Sets the fanout limit guideline for the current project. For more information about fanout limits, see The syn_maxfan Attribute in Microchip Designs , on page 804 .
-globalthreshold value	<i>PolarFire, SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOE, and ProASIC3/3E/3L families.</i> Sets the minimum fanout load value. For more information, see Promote Global Buffer Threshold , on page 807 .
-clock_globalthreshold value	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Sets fanout threshold for clock nets to infer CLKINT. Default value is 2.

Option	Description
-async_globalthreshold value	Sets fanout threshold for asynchronous reset/set nets to infer CLKINT. Default value is 8 for <i>RTG4</i> and 800 for <i>PolarFire</i> , <i>SmartFusion2</i> and <i>IGLOO2</i> .
-opcond value	<p><i>PolarFire</i>, <i>SmartFusion</i>, <i>Fusion</i>, <i>IGLOO2</i>, <i>IGLOO+ / IGLOO / IGLOOe</i>, <i>ProASIC (500K)</i>, <i>ProASICPLUS (PA)</i>, and <i>ProASIC3 / 3E / 3L families</i></p> <p>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 811 for more information.</p>
-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.
-rw_check_on_ram 1 0	<p>Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks.</p> <p>For more information about using this option in conjunction with the <i>syn_ramstyle</i> attribute, see syn_ramstyle, on page 509.</p>
-update_models_cp 1 0	<p><i>PolarFire</i>, <i>SmartFusion</i>, <i>Fusion</i>, <i>IGLOO2</i>, <i>IGLOO+ / IGLOO / IGLOOe</i>, <i>ProASIC (500K)</i>, <i>ProASICPLUS (PA)</i>, and <i>ProASIC3 / 3E / 3L</i></p> <p>When set to 1, the 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. See Update Compile Point Timing Data Option, on page 809, for details.</p>
-low_power_ram_decomp 0 1	<p><i>PolarFire</i>, <i>SmartFusion2</i>, <i>IGLOO2</i>, <i>RTG4</i></p> <p>Enables use of BLK pins of the RAM for reducing power consumption, by fracturing wide RAMs on the address width. Default value is 0.</p>

Option	Description
-seqshift_to_uram 0 1	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Enables inference of URAM if the threshold is met. Default value is 1.
-disable_ramindex 0 1	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Disables the generation of RAMINDEX for RAM blocks, when set to 1.
-microsemi_enhanced_fow 0 1	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Enables advanced constraint writer flow and writes the forward annotation constraints in Libero enhanced constraints format, when the value is set to 1. Default value is 1.
-rep_clkint_driver 0 1	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Enables replication of the register driving a CLKINT as well as some other loads, for which the fanout threshold is not met. Default value is 1.
-ternary_adder_decomp value	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Enables ternary adder implementation with the limit of the adder output width set by default to 66. Ternary adder implementation can be turned off by setting the value to 0.
pack_uram_addr_reg 0 1	<i>PolarFire, SmartFusion2, IGLOO2, RTG4</i> Disables the packing of the read address register in the URAM when the value is set to 0. Default value is 1.
polarfire_ram_init 0 1	<i>PolarFire</i> Disables the RAM initialization on PolarFire devices, when the value is set to 0. Default value is 1.
-gclkint_threshold {}	<i>PolarFire</i> Sets threshold for GCLKINT inference fanout. Default value is 1000.
-rgclkint_threshold {}	<i>PolarFire</i> Sets threshold for RGCLKINT inference fanout. Default value is 100.
-gclk_resource_count {}	<i>PolarFire</i> Sets limit on GCLKINT and RGCLKINT inference. Default value is 24.

Option	Description
-low_power_gated_clock {1/0}	<i>PolarFire</i> Enables inference of clock gating macros, if set to 1. Default value is 0.
-clkint_rgclkint_limit {}	<i>PolarFire</i> Sets limit on the number of RGCLKINTs inferred per CLKINT. Default value is 1.

Microchip Output Files and Forward Annotation

The following procedures show you how to pass information or files that forward annotate information to the Microchip place-and-route tool. This section describes the following:

- [VM Flow Support](#), on page 818
- [Forward-annotating Constraints for Placement and Routing](#), on page 820
- [Specifying Pin Locations](#), on page 821
- [Specifying Locations for Microchip Bus Ports](#), on page 822
- [Specifying Macro and Register Placement](#), on page 823
- [Synthesis Reports](#), on page 823

After synthesis, the software generates a log file and output files for forward annotation to the Microchip P&R tool as described in some of the reports.

VM Flow Support

The tool generates a Verilog output netlist (.vm) for PolarFire, SmartFusion2, RTG4 and IGLOO2 devices for the P&R flow. After synthesis, the tool:

- Writes a separate SDC file (*_vm.sdc).
- Writes a separate TCL file (*_partition_vm.tcl) to forward annotate the timestamps on instances in an incremental compile point flow.

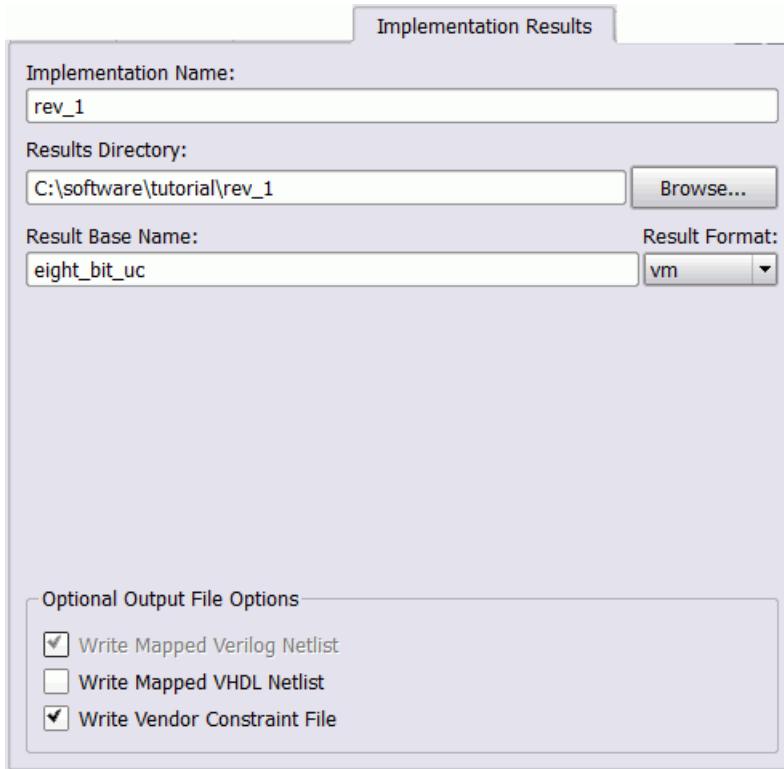
- Forward annotates properties like RTL attributes in the .vm netlist and constraints in an SDC file.

By default, the tool generates an .vm netlist. You can change the netlist from Verilog to EDIF.

The tool now supports the Libero Enhanced constraint flow by default. To disable this flow, following switch needs to be added in the Synplify Pro project .prj file:

```
set_option -Microchip_enhanced_flow 0
```

To select a Verilog output netlist, go to Implementation Options->Implementation Results->Result Format. Select vm from the drop-down menu, click OK and save the project.



Forward-annotating Constraints for Placement and Routing

For Microchip Fusion, IGLOO, ProASIC3, and SmartFusion technology families, the synthesis tool forward annotates timing constraints to placement and routing through an Microchip 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 Microchip constraint file is generated using synthesis tool attributes and constraints.

By default, Microchip 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 Microchip's place-and-route tools has an *_vm.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 Microchip P&R tool.

Synthesis	Microchip
create_clock	<code>create_clock</code> The <code>create_clock</code> constraint is allowed for all NGT families. No wildcards are accepted. The pin or port must exist in the design. The <code>-name</code> argument is not supported as that would define a virtual clock. However, for backward compatibility, a <code>-name</code> argument does not generate an error or warning when encountered in an <i>.sdc</i> file.

Synthesis	Microchip
<code>set_max_delay</code>	<code>set_max_delay</code> The <code>set_max_delay</code> constraint is allowed for all NGT families. Wildcards are accepted.
<code>set_multicycle_path</code>	<code>set_multicycle_path</code> You must specify at least one of the <code>-from</code> or <code>-to</code> arguments, however, it is best to specify both. Wildcards are accepted. Multicycle constraints with <code>-from</code> and/or <code>-to</code> arguments only are supported for Microchip ProASIC3/3E technologies. Multicycle constraints with a <code>-through</code> argument are not supported for any NGT family.
<code>set_false_path</code>	<code>set_false_path</code> Only false path constraints with a <code>-through</code> argument are supported for NGT families. False path constraints with either <code>-from</code> and/or <code>-to</code> arguments are not supported for any NGT family. Wildcards are accepted.

Specifying Pin Locations

In certain technologies you can specify pin locations that are forward-annotated to the corresponding place-and-route tool. The following procedure shows you how to specify the appropriate attributes. For information about other placement properties, see *Specifying Macro and Register Placement*, on page 823.

1. Start with a design using an appropriate Microchip technology.
2. Add the appropriate attribute to the port. For a bus, list all the bus pins, separated by commas. To specify Microchip bus port locations, see *Specifying Locations for Microchip Bus Ports*, on page 822.
 - To add the attribute from the SCOPE interface, click the Attributes tab and specify the appropriate attribute and value.
 - To add the attribute in the source files, use the appropriate attribute and syntax. For details about the attributes in the tables, see the *Attribute Reference Manual*.

Vendor Family	Attribute and Value
Microchip	syn_loc {pin_number} or alspin {pin_number}

Specifying Locations for Microchip Bus Ports

You can specify pin locations for Microchip bus ports. To assign pin numbers to a bus port, or to a single- or multiple-bit slice of a bus port, do the following:

1. Open the constraint file and add these attributes to the design.
2. Specify the syn_noarrayports attribute globally to bit blast all bus ports in the design.

```
define_global_attribute syn_noarrayports {1};
```

3. Use the alspin attribute to specify pin locations for individual bus bits. This example shows locations specified for individual bits of bus ADDRESS0.

```
define_attribute {ADDRESS0[4]} alspin {26}
define_attribute {ADDRESS0[3]} alspin {30}
define_attribute {ADDRESS0[2]} alspin {33}
define_attribute {ADDRESS0[1]} alspin {38}
define_attribute {ADDRESS0[0]} alspin {40}
```

The software forward-annotates these pin locations to the place-and-route software.

Specifying Macro and Register Placement

You can use attributes to specify macro and register placement in Microchip designs. The information here supplements the pin placement information described in [Specifying Pin Locations, on page 821](#) and bus pin placement information described in [Specifying Locations for Microchip Bus Ports, on page 822](#).

For ...	Use ...
Relative placement of Microchip macros and IP blocks	<code>alsloc</code> <code>define_attribute {u1} alsloc {R15C6}</code>

Synthesis Reports

The synthesis tool generates a resource usage report, a timing report, and a net buffering report for the Microchip designs that you synthesize. To view the synthesis reports, click [View Log](#).

Integration with Microchip Tools and Flows

The following procedures provide Microchip-specific design tips.

- [Compile Point Synthesis](#), on page 824
- [Incremental Synthesis Flow](#), on page 825
- [Using Predefined Microchip Black Boxes](#), on page 825
- [Using Smartgen Macros](#), on page 826
- [Microchip Place-and-Route Tools](#), on page 827

Compile Point Synthesis

Microchip PolarFire, SmartFusion, Fusion, IGLOO, Axcelerator, ProASIC3, ProASIC (500K), and ProASICPLUS (PA) Technology Families

Compile-point synthesis is available 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 626 for a description, and [Working with Compile Points](#), on page 605 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 606, and [Update Compile Point Timing Data Option](#), on page 809.)

Incremental Synthesis Flow

Microchip IGLOO2, SmartFusion2, and RTG4 Technologies

The synthesis tool provides timestamps for each manual compile point in the *_partition.tcl file. You can use the timestamps to check whether the compile point was resynthesized in an incremental run of the tool.

To run this flow:

1. Define compile point constraint on the modules in the design. For example:

```
define_compile_point {viewName} -type {locked, partition}  
-cpfile {fileName}
```

2. Run the standard synthesis flow. The synthesis tool writes the timestamps for each compile point in the *designName_partition.tcl* file. For example:

```
set_partition_info -name partitionName -timestamp timestamp
```

For an incremental synthesis run, only affected compile points display new timestamps, while unaffected compile points retain the same timestamps.

Check the Compile Point Summary report available in the log file.

Using Predefined Microchip Black Boxes

The Microchip macro libraries contain predefined black boxes for Microchip macros so that you can manually instantiate them in your design. For information about using ACTGen macros, see [Using Smartgen Macros, on page 826](#). For general information about working with black boxes, see [Defining Black Boxes for Synthesis, on page 514](#).

To instantiate an Microchip macro, use the following procedure.

1. Locate the Microchip macro library file appropriate to your technology and language (v or vhdl) in one of these subdirectories under *installDirectory/lib*.

proasic	ProASIC (PA), ProASIC ^{PLUS} , ProASIC3/3E/3L, Fusion/SmartFusion, and IGLOO/IGLOO+/IGLOOe macros
Microchip	Macros for all other Microchip technologies.

Use the macro file that corresponds to your target architecture. If you are targeting the 1200XL architecture, use the `act2.v` or `act2.vhd` macro library.

2. Add the Microchip macro library *at the top* of the source file list for your synthesis project. Make sure that the library file is first in the list.
3. For VHDL, also add the appropriate library and use clauses to the top of the files that instantiate the macros:

```
library family;
use family.components.all;
```

Specify the appropriate technology in *family*; for example, `act3`.

Using Smartgen Macros

The Smartgen macros replace the ACTgen macros, which were available in the previous Designer 6.x place-and-route tool. The following procedure shows you how to include Smartgen macros in your design. For information about using Microchip macro libraries, see [Using Predefined Microchip Black Boxes, on page 825](#). For general information about working with black boxes, see [Defining Black Boxes for Synthesis, on page 514](#).

1. In Smartgen, generate the function you want to include.
2. For Verilog macros, do the following:
 - Include the appropriate Microchip macro library file for your target architecture in your the source files list for your project.
 - Include the Verilog version of the Smartgen result in your source file list. Make sure that the Microchip macro library is first in the source files list, followed by the Smartgen Verilog files, followed by the other source files.
3. Synthesize your design as usual.

Microchip 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 1102](#) in the *User Guide*.

For details about the place-and-route tools, refer to the Microchip documentation.

Microchip Attribute and Directive Summary

The following table summarizes the synthesis and Microchip-specific attributes and directives available with the Microchip technology. Complete descriptions and examples are in [Summary of Attributes and Directives, on page 19](#).

Attribute/Directive	Description
<code>alsloc</code>	Forward annotates the relative placements of macros and IP blocks to Microchip Designer. This attribute does not apply to ProASIC (500K) and ProASIC ^{PLUS} (PA) designs.
<code>alspin</code>	Assigns scalar or bus ports to Microchip I/O pin numbers. This attribute does not apply to ProASIC (500K) and ProASIC ^{PLUS} (PA) designs.
<code>alspreserve</code>	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.
<code>black_box_pad_pin</code> (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.
<code>black_box_tri_pins</code> (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.
<code>full_case</code> (D)	Specifies that a Verilog case statement has covered all possible cases.
<code>loop_limit</code> (D)	Specifies a loop iteration limit for for loops.
<code>parallel_case</code> (D)	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
<code>syn_allow_retiming</code>	Specifies whether registers can be moved during retiming.
<code>syn_black_box</code> (D)	Defines a black box for synthesis.

(D) indicates directives; all others are attributes.

Attribute/Directive	Description
<code>syn_direct_enable</code>	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.
<code>syn_encoding</code>	Specifies the encoding style for state machines.
<code>syn_enum_encoding (D)</code>	Specifies the encoding style for enumerated types (VHDL only).
<code>syn_global_buffers</code>	Sets the number of global buffers to use in a ProASIC3/3E/3L.
<code>syn_hier</code>	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
<code>syn_insert_buffer</code>	Inserts a clock buffer according to the specified value.
<code>syn_highrel_ioconnector</code>	<i>Synplify Premier</i> Identifies I/O connectors that interface to modules with distributed TMR or DWC to ensure their inputs and outputs are not a single point of failure.
<code>syn_insert_pad</code>	Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.
<code>syn_isclock (D)</code>	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
<code>syn_keep (D)</code>	Prevents the internal signal from being removed during synthesis and optimization.
<code>syn_looplimit</code>	Specifies a loop iteration limit for while loops in the design.
<code>syn_maxfan</code>	Overrides the default fanout guide for an individual input port, net, or register output.
<code>syn_multstyle</code>	Determines how multipliers are implemented for Microchip devices.
<code>syn_netlist_hierarchy</code>	Determines whether the EDIF output netlist is flat or hierarchical.

(D) indicates directives; all others are attributes.

Attribute/Directive	Description
<code>syn_noarrayports</code>	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.
<code>syn_noclockbuf</code>	Turns off the automatic insertion of clock buffers.
<code>syn_noprune</code> (D)	Controls the automatic removal of instances that have outputs that are not driven.
<code>syn_no_compile_point</code>	Use this attribute with the Automatic Compile Point (ACP) feature. If you do not want the software to create a compile point for a particular view or module, then apply this attribute.
<code>syn_pad_type</code>	Specifies an I/O buffer standard.
<code>syn_preserve</code> (D)	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
<code>syn_preserve_sr_priority</code>	Forces set/reset flip-flops to honor the coded priority for the set or reset. ACT 1 and 40MX architectures only.
<code>syn_probe</code>	Adds probe points for testing and debugging.
<code>syn_radhardlevel</code>	Specifies the radiation-resistant design technique to apply to a module, architecture, or register.
<code>syn_ramstyle</code>	Specifies the implementation to use for an inferred RAM. You apply <code>syn_ramstyle</code> globally, to a module, or to a RAM instance.
<code>syn_reference_clock</code>	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
<code>syn_replicate</code>	Controls replication.
<code>syn_resources</code>	Specifies resources used in black boxes.
<code>syn_safe_case</code>	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.
<code>syn_sharing</code> (D)	Specifies resource sharing of operators.

(D) indicates directives; all others are attributes.

Attribute/Directive	Description
<code>syn_shift_resetphase</code>	Allows you to remove the flip-flop on the inactive clock edge, built by the reset recovery logic for an FSM when a single event upset (SEU) fault occurs.
<code>syn_state_machine</code> (D)	Determines if the FSM Compiler extracts a structure as a state machine.
<code>syn_tco< n ></code> (D)	Defines timing clock to output delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tpd< n ></code> (D)	Specifies timing propagation for combinational delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tristate</code> (D)	Specifies that a black-box pin is a tristate pin.
<code>syn_tsu< n ></code> (D)	Specifies the timing setup delay for input pins, relative to the clock. The <i>n</i> indicates a value between 1 and 10.
<code>syn_useenables</code>	Generates clock enable pins for registers.
<code>syn_vote_loops</code>	<i>Synplify Premier</i> Specifies whether or not to add any synchronous voter to a sequential feedback path for a distributed TMR module.
<code>syn_vote_register</code>	<i>Synplify Premier</i> Enables voter logic to be inserted after every register of the module specified for distributed TMR.
<code>translate_off/translate_on</code> (D)	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

(D) indicates directives; all others are attributes.

APPENDIX E

Designing with Xilinx

The following topics describe how to design and synthesize with different Xilinx technologies:

- [Basic Support for Xilinx Designs](#), on page 834
- [Xilinx Components](#), on page 839
- [Xilinx Constraints, Attributes, and Options](#), on page 873
- [Optimizing Xilinx Designs](#), on page 879
- [Xilinx Optimizations](#), on page 916
- [Xilinx Device Mapping Options](#), on page 925
- [Xilinx Output Files and Forward Annotation](#), on page 941
- [Integration with Xilinx Tools and Flows](#), on page 947
- [Xilinx Attribute and Directive Summary](#), on page 951

Basic Support for Xilinx Designs

This section describes general guidelines for using the synthesis tool and Xilinx place-and-route (P&R) tool with Xilinx devices. Refer to:

- [Xilinx Device Support](#), on page 834
- [Netlist Format](#), on page 834
- [Xilinx Stacked Silicon Interconnect \(SSI\) Technology](#), on page 837
- [Supported Synthesis Design Flows for Xilinx Designs](#), on page 838

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.

See the following for device-specific information:

- [Virtex and Spartan-II and Later Technologies](#), on page 925
- [Xilinx CPLDs](#), on page 935

After synthesis, the tool generates an EDIF or VM netlist that is used as input to run the Xilinx Vivado place-and-route tool. Also, the tool can generate XNF or EDIF netlists that are used as input to run the Xilinx ISE Design Suite.

Netlist Format

The synthesis tool outputs EDIF, VM, or XNF netlist files for use with the Xilinx P&R application. These files have .edf, .vm, and .xnf extensions.

On the Implementation Results tab of the Implementation Options dialog box, the following file formats: edif, vm, and xnf 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/vm/xnf
```

Targeting Output for Xilinx

You can generate output targeted for Xilinx.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

The following table summarizes the outputs to set for the different devices, and shows the P&R tools for which the output is intended.

Vendor Support	Output Netlist	P&R Tool
Xilinx 7 Series and UltraScale families	EDIF/VM (.edf or .vm)	Vivado
Xilinx Virtex-6 and Spartan-6 and earlier families	EDIF (.edf)	Design Manager or ISE Project Navigator
Xilinx CoolRunner families	EDIF (.edf) or XNF (.xnf)	Design Manager or ISE Project Navigator

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

Customizing Netlist Formats

The following table lists some attributes for customizing your Xilinx output netlists:

For ...	Use ...
Netlist formatting	<code>syn_netlist_hierarchy</code> <code>define_global_attribute syn_netlist_hierarchy {0}</code>
EDIF formatting	<code>syn_edif_bit_format</code> <code>define_global_attribute syn_edif_bit_format {%n<%i>}</code>

For ...	Use ...
	<i>syn_edif_bit_format</i> define_global_attribute syn_edif_name_length {restricted}
	<i>syn_edif_scalar_format</i> define_global_attribute syn_edif_scalar_format {%u}
Bus specification	<i>syn_noarrayports</i> define_global_attribute syn_noarrayports {1}

Xilinx Forward Annotation

The synthesis tool generates Xilinx-compliant constraint files from selected constraints that are forward-annotated (read in and then used) by the Xilinx Vivado or ISE place-and-route software. The Xilinx constraint file uses the `xdc` extension for Vivado. The `ncd`, `ncf`, or `ucf` extension is used for ISE. This constraint file must be imported into the Xilinx flow.

By default, Xilinx 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).

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 133](#).

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 [Design Planner - Working with Xilinx SSI Devices, on page 1028](#).

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
```

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 38
Logic synthesis with floorplan	(<i>Synplify Premier with Design Planner</i>). See Design Plan-Based Logic Synthesis, on page 40 in the <i>User Guide</i> .
Logic synthesis with a synthesis strategy	(<i>Synplify Premier</i>). See Setting Synthesis Strategies, on page 595 in the <i>User Guide</i> .
Design Plan-Based logic synthesis	(<i>Synplify Premier with Design Planner</i>). See Design Plan-Based Logic Synthesis, on page 40 in the <i>User Guide</i> .

For other flows to integrate with Xilinx software, see [Integration with Xilinx Tools and Flows, on page 947](#) for information.

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:

- [DSP Components in Virtex Designs](#), on page 839
- [DSP48 Block Inference](#), on page 840
- [DSP48 Symmetric Rounding for XNOR Inferencing](#), on page 847
- [Xilinx RAM Inference](#), on page 854
- [Xilinx ROM Inference](#), on page 857
- [Asynchronous Registers](#), on page 859
- [DDR Registers](#), on page 860
- [Dynamic Shift Register Lookup \(SRL\) Table](#), on page 864
- [Latch Mapping](#), on page 866
- [Xilinx I/O Support](#), on page 866

DSP Components in Virtex Designs

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.

DSP Examples

The following examples infer an adder to a DSP:

Verilog	<code>wire [9:0] adder_sig /* synthesis syn_DSPStyle = "dsp48" */;</code>
VHDL	<code>attribute syn_DSPStyle : string;</code> <code>attribute syn_DSPStyle of adder_sig : signal is "dsp48";</code>

DSP Examples

The following examples infer a multiplier to logic:

```
Verilog    wire [9:0] mult_sig /* synthesis syn_DSPstyle = "logic" */;
```

```
VHDL      attribute syn_DSPstyle : string;
           attribute syn_DSPstyle of mult_sig : signal is "logic";
```

DSP48 Block Inference

Xilinx Virtex-6 and Newer Technologies

The synthesis tool can map the following multi-bit vector logic functions to DSP48 blocks:

- AND ($a \& b$) and NAND $\sim(a \& b)$
- OR ($a | b$) and NOR $\sim(a | b)$
- XOR ($a ^ b$) and XNOR $\sim(a ^ b)$

Logic operations on multi-bit vectors can be mapped to DSP48 blocks with the following conditions:

- The input vectors to the logic operations and the output result of the logic operations must be registered.
- Logic operations less than 5 bits are not mapped to DSP48 blocks.
- Logic operations on bit vectors of 5 bits up to 48 bits can be mapped to one DSP48 block.
- Logic operations on bit vectors of 49 bits up to 52 bits are mapped to one DSP48, with the upper bits mapped to LUTs. The number of LUTs is based on the number of bits modulo 48. For example:
 - 49 bit vectors are mapped to one DSP and one LUT.
 - 50 bit vectors are mapped to one DSP and two LUTs.
 - 51 bit vectors are mapped to one DSP and three LUTs.
 - 52 bit vectors are mapped to one DSP and four LUTs.
- Logic operations on bit vectors from 53 bits to 96 bits are mapped to two DPS48 blocks. For logic operations on bit vectors from 97 bits to 100 bits, the logic is mapped to two DSP48 blocks and the remaining

bits are mapped to LUTs (modulo 48). This process repeats for multi-bit vectors that are multiples of 48.

- Reduction AND (`& a`) and NAND `~(& a)` operations
 - These operations can be mapped to DSP48 blocks, if their bus widths are from 5 bits up to 96 bits. Designs with bits greater than 96 bits cannot be mapped to the DSP48 block.
 - For reduction NAND operations, the inverter remains outside of the DSP48 block.

You must apply the `syn_DSPstyle` attribute value of `dsp48` on the register following the logic operand, in order to infer the DSP48 block. This attribute can be specified either in the RTL on the register or in an FDC constraint file. For details about the attribute syntax, see [syn_DSPstyle, on page 219](#).

Effects of Using `syn_DSPstyle`

Here are examples that show how DSP48 blocks are inferred using the `syn_DSPstyle` attribute.

Example 1: Multi-bit AND Operation

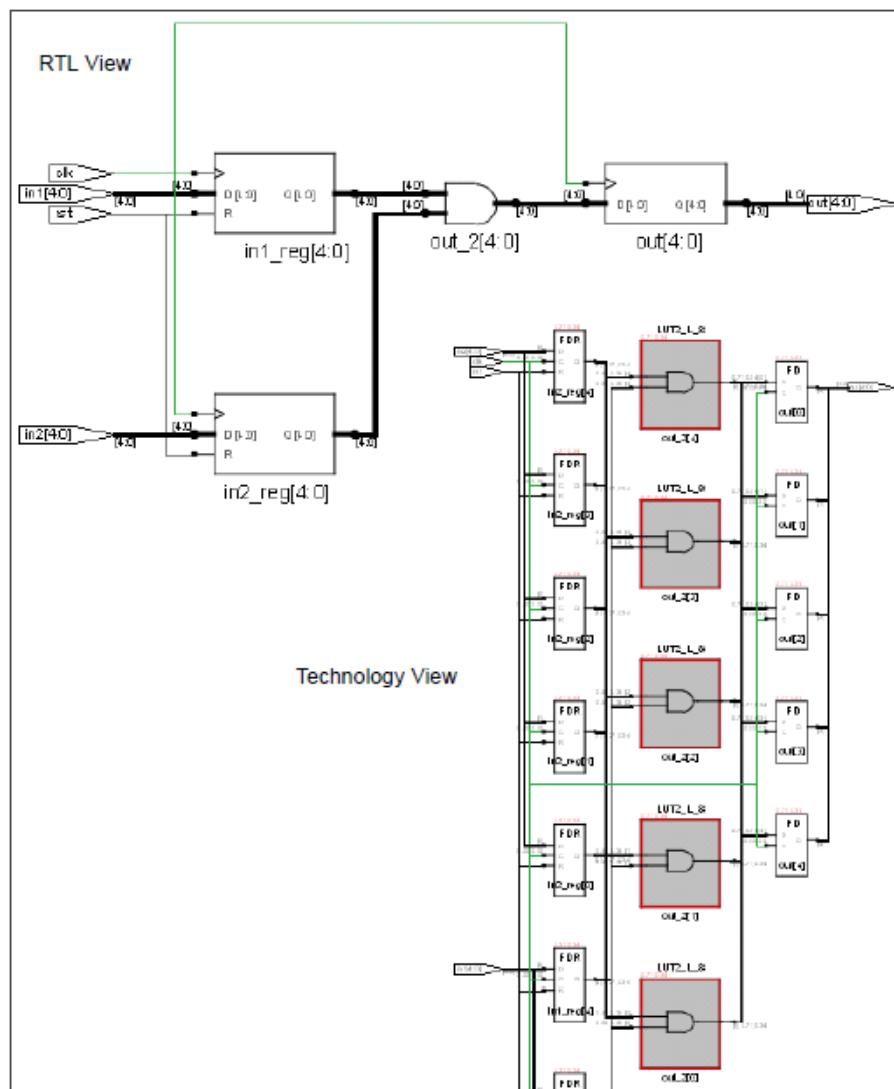
```
module top (
    in1,
    in2,
    out,
    clk,
    rst
);

parameter DWIDTH = 5;

input [DWIDTH-1:0]in1;
input [DWIDTH-1:0]in2;
input clk;
```

```
input rst;  
output [DWIDTH-1:0] out;  
  
reg [DWIDTH-1:0] in1_reg;  
reg [DWIDTH-1:0] in2_reg;  
reg [DWIDTH-1:0] out;  
  
always @ (posedge clk)  
if (rst)  
    in1_reg <= 0;  
else  
    in1_reg <= in1;  
  
always @ (posedge clk)  
if (rst)  
    in2_reg <= 0;  
else  
    in2_reg <= in2;  
always @ (posedge clk)  
    out <= (in1_reg & in2_reg);  
  
endmodule
```

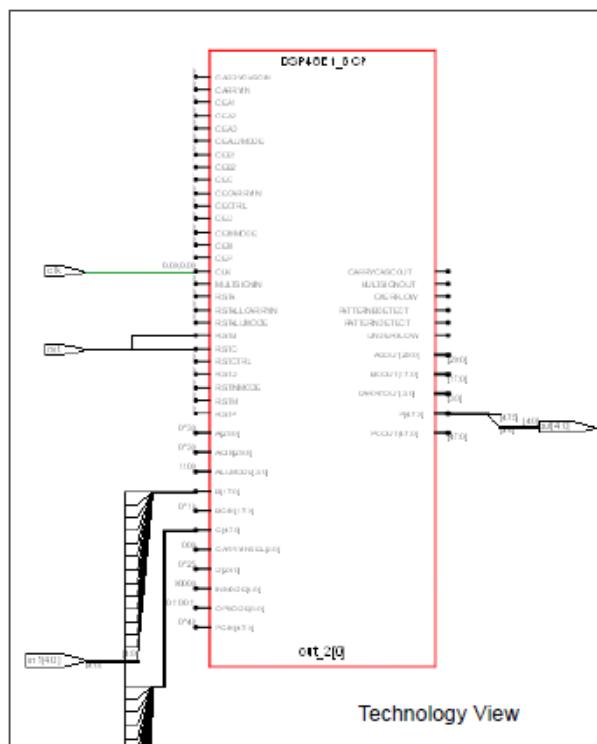
Without applying the attribute, the 5-bit AND operation does not get mapped to the DSP48 block.



Make sure to add a register at the inputs and apply the following attribute to the output register in the FDC constraint file:

```
define_attribute {i:out} {syn_dspstyle} {dsp48}
```

The DSP48 block is inferred as shown in the Technology view.



Example 2: Multi-bit Reduced AND Operation

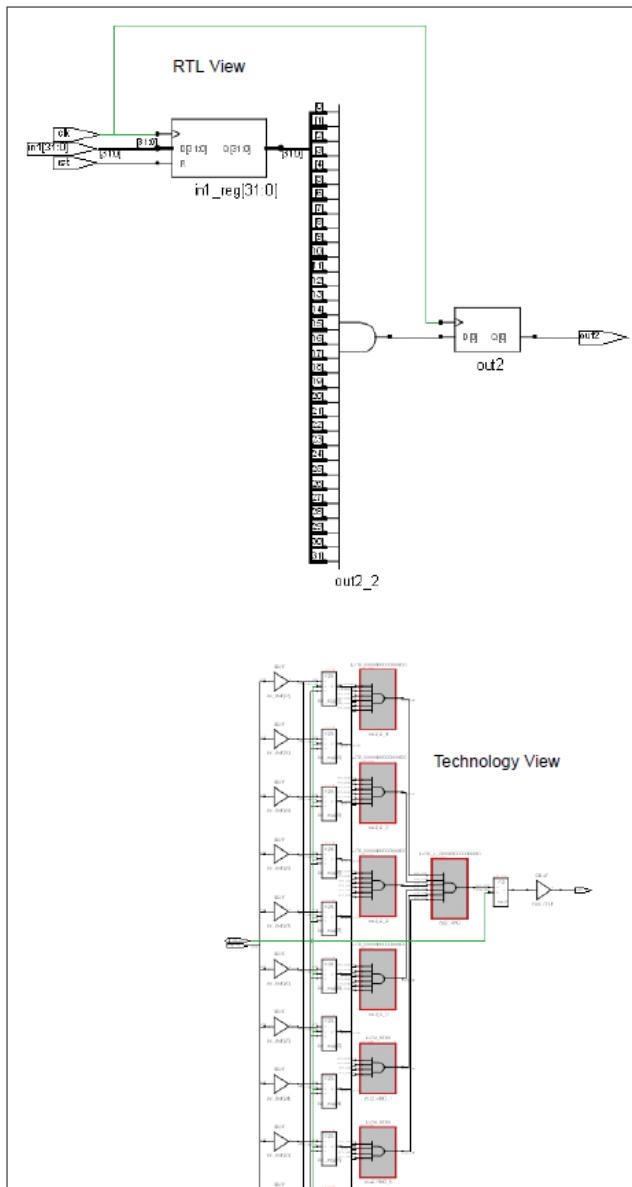
```
module top ()  
    input in1,  
    output out2,  
    input clk,  
    input rst  
);
```

```
parameter DWIDTH = 32;
```

```
input [DWIDTH-1:0]in1;
```

```
input clk;  
input rst;  
output out2;  
  
reg [DWIDTH-1:0]in1_reg;  
reg out2 /* synthesis syn_dspstyle = "dsp48" */;  
  
always @ (posedge clk)  
if (rst)  
    in1_reg <= 32'd0;  
else  
    in1_reg <= in1;  
  
always @ (posedge clk)  
out2 <= (& in1_reg);  
  
endmodule
```

Without applying the attribute, the 32-bit reduced AND operation does not get mapped to the DSP48 block.



Make sure to add a register to the inputs and apply the following attribute on the output register in the RTL:

```
reg out2 /* synthesis syn_DSPstyle = "dsp48" */;
```

The DSP48 block is inferred in the Technology view.

DSP48 Symmetric Rounding for XNOR Inferencing

Xilinx Virtex-5 and Newer Technologies

The synthesis tool can map the XNOR logic of the multiplier input to the DSP48 block. Xilinx DSP48 architecture supports symmetric rounding, where the MSB of the signed multiplier is XNORed ($A[24] \text{xnor } B[17]$). Optionally, the XNOR output can be registered to match the MREG pipeline delay. When the XNOR is packed, the CARRYINSEL of the DSP48 block gets tied to 110.

Example: Infers XNOR and Multiplier to the DSP48 Block

```
module test (clk, rst, en, a_ip, b_ip, c_op);
parameter a_width=25;
parameter b_width=18;
parameter mult_width=a_width+b_width;
parameter op_width=33;
parameter round_value = 511;

////***** Inputs declaration
input clk;
input rst;
input en;
input signed [a_width - 1 : 0] a_ip;
input signed [b_width - 1 : 0] b_ip;

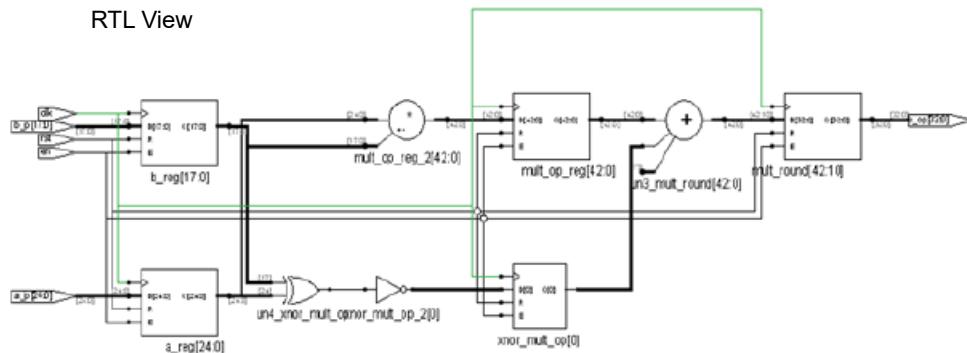
////***** Outputs declaration
output signed [op_width - 1 : 0] c_op;

reg signed [a_width-1:0] a_reg;
reg signed [b_width-1:0] b_reg;
reg [0:0] xnor_mult_op;
reg signed [mult_width-1:0] mult_op_reg;
reg signed [mult_width-1:0] mult_round;
```

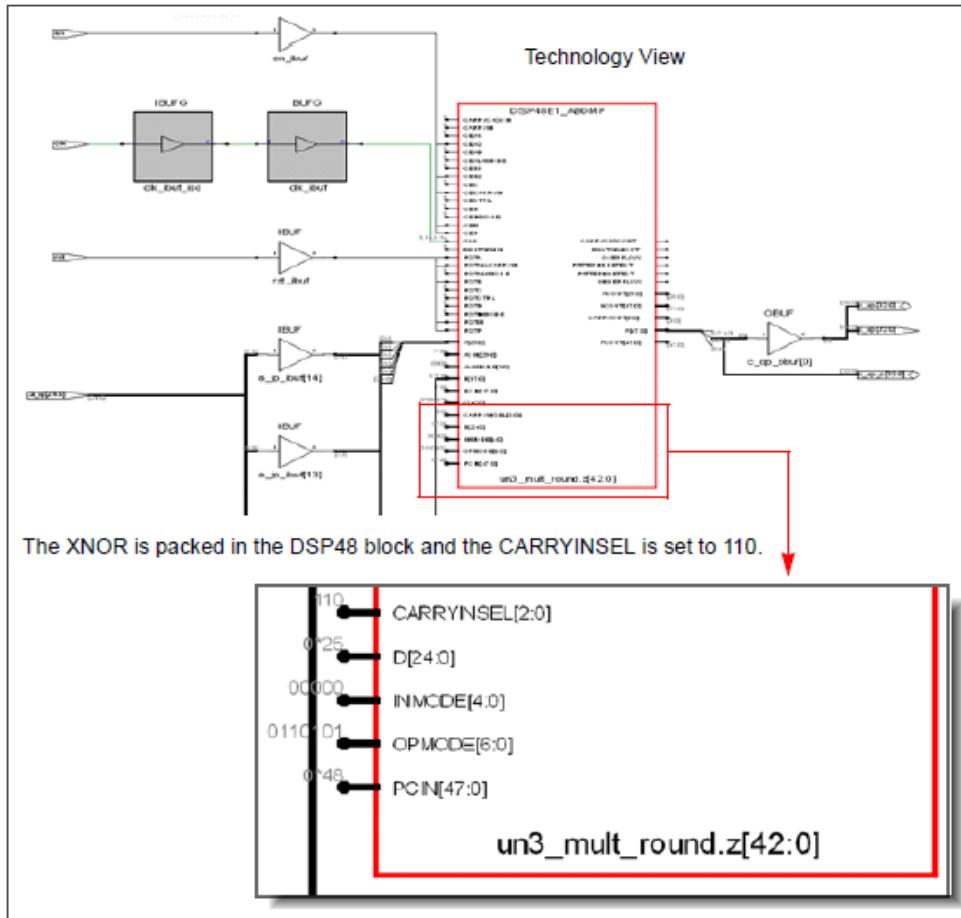
```
always@ (posedge clk)
begin
    if (rst)
        begin
            a_reg <= {a_width{1'b0}};
            b_reg <= {b_width{1'b0}};
        end
    else
        begin
            if (en)
                begin
                    a_reg <= a_ip;
                    b_reg <= b_ip;
                end
            end
        end
    end

always@ (posedge clk)
begin
    if (rst)
        begin
            xnor_mult_op <= 0;
            mult_op_reg <= 0;
            mult_round <= 0;
        end
    else
        begin
            if (en)
                begin
                    xnor_mult_op <= a_reg[a_width-1] ~^ b_reg[b_width-1];
                    mult_op_reg <= a_reg * b_reg;
                    mult_round <= mult_op_reg + round_value + xnor_mult_op;
                end
            end
        end
    assign c_op = mult_round[42:10];
endmodule
```

Here is the RTL view for this example.



The XNOR, registers, and multiplier are mapped to a single DSP48 block shown in the Technology view below.



Dynamic Inmode Support for DSP48 Inferencing

Xilinx Virtex-6 and 7 Series Technologies

The synthesis tool can map the dynamic signals that control the pre-adder or multiplier input for the DSP48 block. The Xilinx DSP48 architecture uses the INMODE[4:0] ports of the DSP block to support the dynamic signals for the pre-add or pre-sub, and other functions defined by the architecture.

Example: Dynamic Inmode for the Inferred DSP48 Block

```

module test (clk,rst, cen,inmode,a,b,d,dout);

/// parameter definition
parameter a_width=23;
parameter b_width=18;
parameter c_width=43;
parameter op_width=43;
parameter m_width= a_width + b_width + 1;
parameter d_width=23;
parameter ad_width=24;

////***** Inputs declaration
input clk;
input [4:0]inmode;
input rst;
input cen;
input signed [a_width - 1:0] a;
input signed [b_width - 1:0] b;
input signed [d_width - 1:0] d;

////***** Outputs declaration
output signed [op_width - 1:0] dout;
reg signed [a_width-1:0] a1_reg;
reg signed [b_width-1:0] b1_reg;
reg signed [a_width-1:0] a2_reg;
reg signed [b_width-1:0] b2_reg;
reg signed [d_width-1:0] d_reg;
reg signed [ad_width-1:0] ad_reg;
reg signed [m_width-1:0] m_reg;

wire signed [a_width-1:0] a1_a2_reg_mux, a_mux_op;
wire signed [b_width-1:0] b1_b2_reg_mux;
wire signed [d_width-1:0] d_mux_op;
wire inmode_0,inmode_1,inmode_2,inmode_3, inmode_4;
wire signed [ad_width-1:0] dyn_preadd_sub;

assign {inmode_4,inmode_3,inmode_2,inmode_1, inmode_0} = inmode;

always@(posedge clk)
begin
    if (rst)
    begin
        a1_reg <= {a_width{1'b0}};
        a2_reg <= {a_width{1'b0}};
        d_reg <= {d_width{1'b0}};
    end
end

```

```

ad_reg    <= {ad_width{1'b0}};
b1_reg    <= {b_width{1'b0}};
b2_reg    <= {b_width{1'b0}};
m_reg     <= {m_width{1'b0}};
end
else
begin
  if (cen)
  begin
    a1_reg <= a;
    a2_reg <= a1_reg;
    d_reg <= d;
    ad_reg <= dyn_preadd_sub;
    b1_reg <= b;
    b2_reg <= b1_reg;
    m_reg <= ad_reg * b1_b2_reg_mux;
  end
end
end

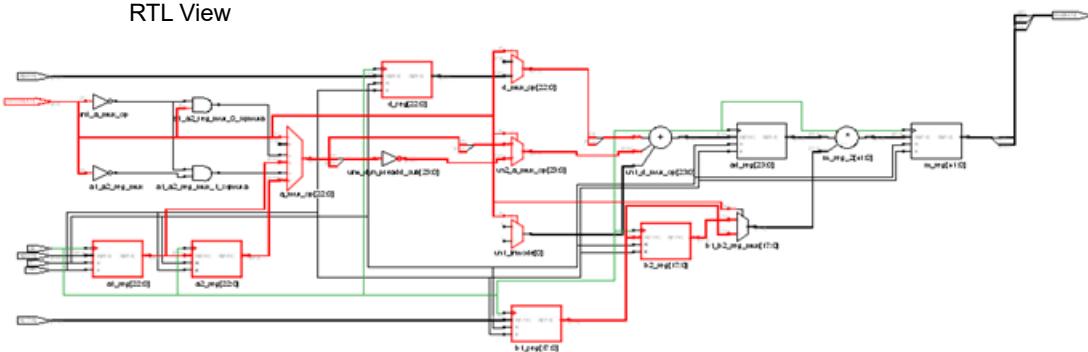
assign a1_a2_reg_mux = inmode_0 ? a1_reg : a2_reg ;
assign a_mux_op = inmode_1 ? {a_width{1'b0}} : a1_a2_reg_mux;
assign d_mux_op = inmode_2 ? d_reg : {d_width{1'b0}} ;
assign dyn_preadd_sub = inmode_3 ? (d_mux_op - a_mux_op) :
(d_mux_op + a_mux_op);
assign b1_b2_reg_mux = inmode_4 ? b1_reg : b2_reg;
assign dout = m_reg;

endmodule

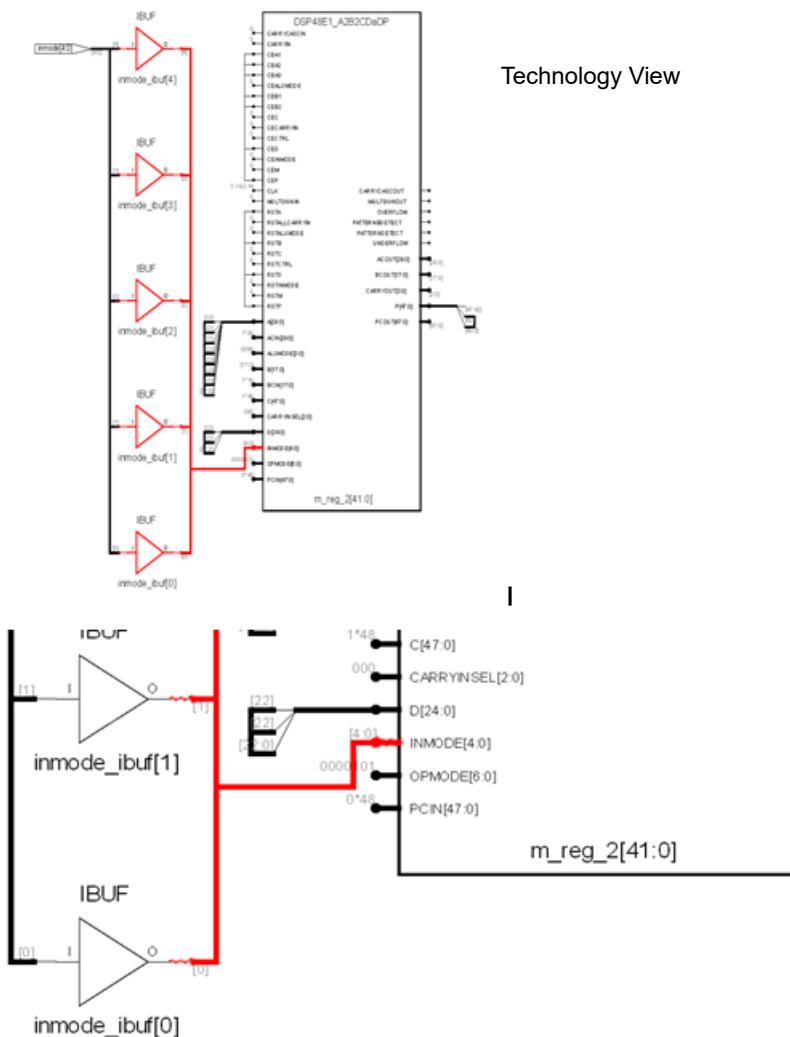
```

Here is the RTL view for this example, showing all the highlighted logic that will be packed into the DSP.

RTL View



The dynamic signals connected to INMODE[4:0] for the inferred DSP block are highlighted in the Technology view below:



Xilinx RAM Inference

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.

- [Mapping of UltraRAM](#), on page 854
- [Cascading Block RAM](#), on page 855
- [Mapping of Asymmetric Ports to Block RAM](#), on page 855
- [Mapping of Byte Enables to Block RAM](#), on page 855
- [Distributed RAM Implementations](#), on page 855
- [Multi-Port RAM Implementations](#), on page 856
- [Single-port and Dual-port Block RAM Implementations](#), on page 856
- [Parity Bus](#), on page 856
- [Disabling RAM Inference](#), on page 857
- [Xilinx Spartan-6 RAMB8BWER Component Limitation](#), on page 857

Mapping of UltraRAM

Xilinx UltraScale+ Technologies

(*Kintex UltraScale+, Virtex UltraScale+, and Zynq UltraScale+*)

The synthesis tools provide support for UltraRAM (URAM) blocks, in addition to the block RAM (RAMB36E2 or RAMB18E2) for Xilinx UltraScale+ devices. UltraRAM is a high-density memory block that has eight times the storage capacity of a block RAM. It consists of 288K bits (4Kx72 bits) of storage with two independent access ports that share a single clock. The tool can automatically infer an UltraRAM block for single-port and simple dual-port RAM. For details, see [UltraRAM Examples](#), on page 305.

Cascading Block RAM

*Xilinx UltraScale, UltraScale+ Technologies
(Kintex UltraScale, UltraScale+, Virtex UltraScale, UltraScale+, and Zynq UltraScale+)*

The synthesis tools provide support for multi-level cascading of block RAM (RAMB36E2) for Xilinx UltraScale+ devices. Multi-level cascading provides QOR improvement. Block RAMs are built from the bottom-up directly in the block ram column. The length of the block RAM chain is controlled by the `syn_bram_cascade_height` attribute. See [syn_bram_cascade_height, on page 157](#) in the *FPGA Synthesis Attribute Reference Manual* for more information.

Mapping of Asymmetric Ports to Block RAM

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.

For details, see [Asymmetric RAM Inference, on page 269](#) in the *FPGA Synthesis User Guide* and [Asymmetric RAM Examples, on page 269](#).

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 [Byte-Enable RAM Inference, on page 302](#) in the *FPGA Synthesis User Guide* and [Byte-Enable RAM Examples, on page 302](#).

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 [Distributed RAM Inference, on page 264](#) in the *FPGA Synthesis User Guide* and [Distributed RAM Examples, on page 267](#).

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 coding information, see [Distributed RAM Inference, on page 264](#).

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.

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.

- 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 [Block RAM Examples, on page 239](#) for more information and examples.

For more information and examples, see [Block RAM Inference, on page 233](#) in the *FPGA Synthesis User Guide* and [Block RAM Examples, on page 239](#).

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 DI[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 509](#).

Xilinx Spartan-6 RAMB8BWER Component Limitation

The Xilinx Spartan-6 RAMB8BWER component that is configured as simple dual-port (SDP) only works reliably with 36 bits on both ports. The RAMB8BWER component might be inferred with different port widths.

As a workaround for instantiated components, you can try the following:

- Change all RAMB8BWER components from SDP mode to True Dual Port (TDP) mode.
- Change all RAMB8BWER components from SDP mode to RAMB16BWER components in SDP mode.
- Pad buses that are less than 36 bits to a full 36 bits for each RAMB8BWER component in SDP mode.

Xilinx ROM Inference

You can map ROM to block RAM with the `syn_romstyle` attribute. For Xilinx architectures, the software can map ROM into block RAM, provided you follow the guidelines in this procedure.

1. Place a `dff` register in front of the ROM, or place one of the following after the ROM:

Asynchronous	Synchronous
dff, dffe	
dffr, dffre	sdffr, sdffre
dffs, dffse	sdffs, sdffse
dffpatr, dffpatre	sdffpatr, sdffpatre

where dffe is an enabled flip-flop, dffre is an enabled flip-flop with asynchronous reset, dffse is an enabled flip-flop with asynchronous set, and dffpatre is an enabled, vectored flip-flop with asynchronous reset pattern.

2. Ensure that registers and ROMs are within the same hierarchy.
3. Make sure that at least half the addresses possess assigned values. For example, in a ROM with ten address bits (1024 unique addresses), at least 512 of those unique addresses must be assigned values.
4. Specify the `syn_romstyle` attribute with the value set to `block_rom`.
5. Synthesize the design.

The software maps the ROM into block RAM.

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 dffpatre. 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 189](#). 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 `sdfppate`. 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 201](#) and [syn_direct_reset, on page 195](#). The compiler does this automatically.

Asynchronous Registers

The synthesis software can synchronize registers in a chain for a signal across two asynchronous clock domains. You can specify the number of synchronization registers in the chain, for which the `ASYNC_REG` attribute is applied. Any number of registers after this specified value can be packed into a SRL. The `ASYNC_REG` attribute is forward annotated to the Xilinx XDC file. This prevents these registers from being mapped to SRL primitives and allows the registers to be placed in the same region.

The tool uses the information in the FDC constraint file that defines these asynchronous clock domains.

- Specify the asynchronous clocks
 - Multiple clocks that are specified as separate and asynchronous clock groups in the constraint file. The software recognizes the launch register and subsequent capture registers clocked in the chain by the clock groups. For example:

```
create_clock -name {clk1} {p:clk1} -period 2.5
create_clock -name {clk2} {p:clk2} -period 3.0
set_clock_groups -asynchronous -group {clk1} -group {clk2}
```

The software interprets that the registers are in separate clock domains either from the fdc constraints or when the Auto Constrain option is enabled for the design.

- Generated or derived (DCM/PLL and ICG) clocks that are specified as separate and asynchronous clock groups in the constraint file. Asynchronous registers are implemented after gated clock conversion occurs.
- Specify the number of registers to be used in the synchronization chain with the `syn_async_reg` attribute. For details about this attribute, see [syn_async_reg, on page 137](#).

- To prevent registers in the synchronization chain from having the ASYNC_REG attribute applied, use the syn_srlstyle attribute on the launch and capture registers.

The synthesis software generates a message in the log file, identifying the registers that are being used for asynchronous cross-clock domains. The message also specifies the find commands you can use to produce a list of the registers in the chain with the ASYNC_REG property.

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 FDDRSE/FDDRCPE primitives or ODDR (Virtex-4/Virtex-5/Virtex-6). Output DDR registers (FDDRSE/FDDRCPE/ODDR) are automatically inferred without having to instantiate them in the HDL. The following examples show the HDL coding styles to infer the output DDR registers. Note, the clock enable signal has priority over the reset and set signals.

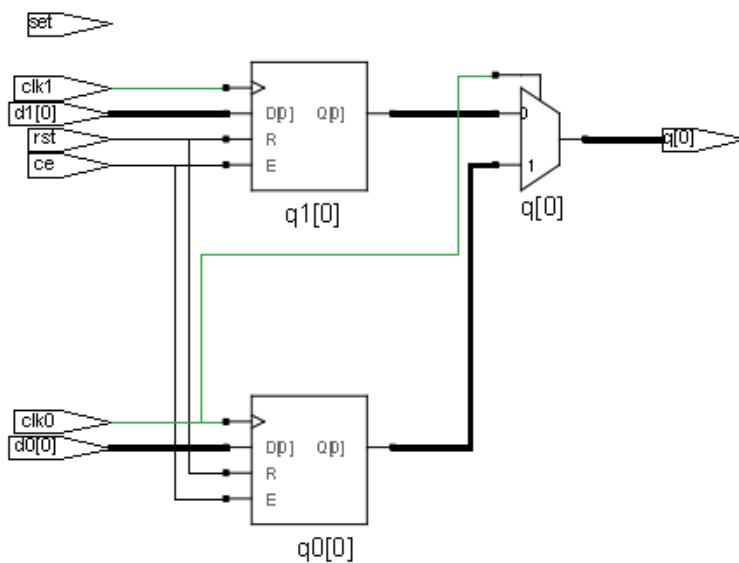
Example 1: Output DDR Registers

```
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(ce)
    q0 = d0;
else if(rst)
    q0 = 1'b0;
else if(set)
    q0 = 1'b1;
end

always @ (posedge clk1)
begin
    if(ce)
        q1 = d1;
    else if(rst)
        q1 = 1'b0;
    else if(set)
        q1 = 1'b1;
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.



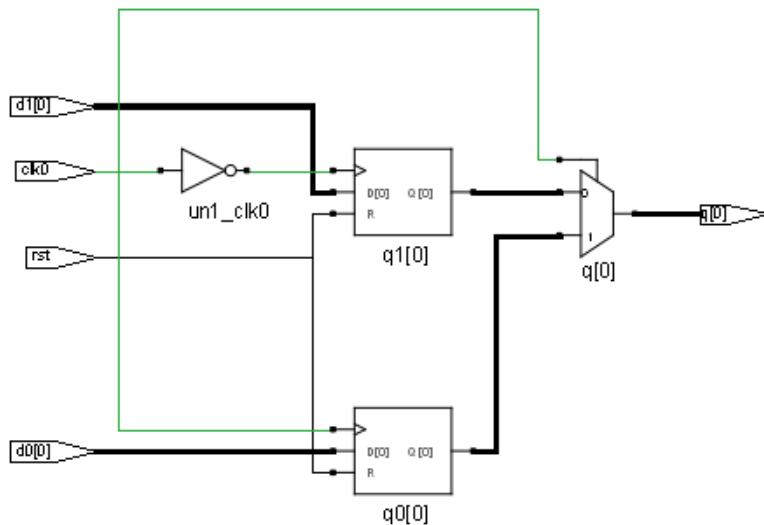
Example 2 - Output DDR Registers (Alternative Coding Style)

```
module fddrrse(d1, d0, clk0, rst, q);
    input [0:0]d1;
    input [0:0]d0;
    input clk0;
    input rst;
    output [0:0]q;
    reg [0:0]q0;
    reg [0:0]q_temp0;
    reg [0:0]q_temp1;
    reg [0:0]q1;
    always @ (posedge clk0)
        begin
            if (rst)

```

```
q0 <= 1'b0;  
else  
    q0 <= d0;  
end  
  
always @ (negedge clk0)  
begin  
    if(rst)  
        q1 <= 1'b0;  
    else  
        q1 <= d1;  
end  
  
assign q = clk0 ? q0 : q1;  
endmodule
```

1. Use positive and negative edges of the same clock to clock the two registers which are then inferred as DDR flip-flops.
2. Use the positive edge of clk0 to trigger q0 and 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 537](#) 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.

```

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.

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 866
- [Xilinx I/O Pads](#), on page 867
- [Xilinx I/O Insertion](#), on page 867
- [Xilinx IOB Packing](#), on page 867
- [Xilinx Differential I/O Buffers](#), on page 867
- [Xilinx Differential I/O Buffers \(7 Series and Newer Technologies\)](#), on page 868
- [Xilinx BUFGMUX](#), on page 868
- [Xilinx I/Os and Pin Locations](#), on page 868
- [Xilinx Regional Clock Buffers](#), on page 868

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 appro-

priate library that defines the macros as black boxes. The macro libraries are located in the *install_dir/lib/xilinx* directory. See [Designing for Xilinx Architectures, on page 879](#).

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 880](#).

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.

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 351](#).

Note: If a BUFG is required when I/O insertion is disabled, replace the IBUF with IBUFG.

Xilinx 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 897](#).

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.

The `syn_diff_io` attribute is supported in the compile point flow.

See [Working with Xilinx Buffers, on page 906](#).

Xilinx Differential I/O Buffers (7 Series and Newer Technologies)

Xilinx 7 Series and UltraScale technologies

By default, the synthesis tool does not automatically infer differential I/O buffers. You can direct the tool to infer IBUFDS, IBUFGDS, OBUFDS, and IOBUFDS by updating the constraint file (fdc) as follows:

- Set the `syn_loc` attribute for the specified signals with location constraints that constitute a differential pair on the part/package.
- Specify the correct IOSTANDARD based on the part/package for these differential pair signals.

Refer to the Xilinx documentation for the part/package specifications.

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 906](#).

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 900](#).

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 907](#).

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 880](#).

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

- \$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
```

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;
```

Xilinx Constraints, Attributes, and Options

Specify timing constraints, general attributes, and Xilinx-specific attributes to improve your design. Manage these files with the SCOPE constraints and attributes editor. This section explains how to implement Xilinx constraints, attributes, and options. Refer to:

- [Xilinx I/O Support](#), on page 866
- [Macros and Black Boxes](#), on page 869
- [Relative Locations](#), on page 877
- [Wide Adders in Virtex-5 and Virtex-6 Designs](#), on page 877
- [Control Sets in Spartan/Virtex Architectures](#), on page 878

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 families, see [Virtex-6 and Spartan-6 I/O Standards](#), on page 875.

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan	Virtex-5	Virtex-II Virtex-IIIP 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+	GTL_P	X	X	X
HSTL_Class_I	HSTL_I	X	X	X

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan	Virtex-5	Virtex-II Virtex-IIIP Virtex-4
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
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

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan	Virtex-5	Virtex-II Virtex-IIP Virtex-4
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
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

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan-6	Virtex-6
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	
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

Synopsis FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan-6	Virtex-6
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 321](#) for a list of industry I/O standards.
- [Xilinx Attribute and Directive Summary, on page 951](#) for a list of Xilinx attributes and directives.

Relative Locations

Use these Xilinx-specific attributes to specify the relative position of components in your designs.

- `xc_map`
- `xc_rloc`
- `xc_usest`

See [Specifying RLOCs, on page 912](#) 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 914](#).

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).

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 219](#)). See [Inferring Wide Adders, on page 883](#) 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 place-and-route tool 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, on page 525](#).

Optimizing Xilinx Designs

This section contains tips for working with Xilinx designs:

- [Designing for Xilinx Architectures](#), on page 879
- [Specifying Xilinx Macros](#), on page 880
- [Specifying Global Sets/Resets and Startup Blocks](#), on page 882
- [Inferring Wide Adders](#), on page 883
- [Instantiating CoreGen Cores](#), on page 886
- [Initializing Xilinx RAM](#), on page 889
- [Specifying Xilinx Register INIT Values](#), on page 895
- [Packing Registers for Xilinx I/Os](#), on page 897
- [Using Clock Buffers in Virtex Designs](#), on page 908
- [Working with Clock Skews in Xilinx Virtex-5 Designs](#), on page 910
- [Instantiating Special I/O Standard Buffers for Virtex](#), on page 911
- [Specifying RLOCs](#), on page 912
- [Specifying RLOCs and RLOC_ORIGINS with the synthesis Attribute](#), on page 914

For additional Xilinx-specific techniques, see [Xilinx Partition Flow](#), on page 1142, [Xilinx Partition Flow for Versions Before ISE 12.1](#), on page 1147, [Automatic RAM Inference](#), on page 229, and [Inferring Shift Registers](#), on page 537. Note that some of these features are not available in the Synplify product.

Designing for Xilinx Architectures

The tips listed here are in addition to the technology-independent design tips described in [Tips for Optimization](#), on page 554.

- For critical paths, attach the `xc_fast` attribute to the I/Os.
- To ensure that frequency constraints from register to output pads are forward annotated to the P&R tools, add default input delay and output delay constraints of 0.0 in the synthesis tool. The synthesis tool forward-

annotates the frequency constraints as PERIOD constraints (register-to-register) and OFFSET constraints (input-to-register and register-to-output). The place-and-route tools use these constraints.

- Run successive place-and-route iterations with progressively tighter timing constraints to get the best results possible.
- When using VHDL, specify a UNISIM library using the following syntax:

```
library unisim;
use unisim.vcomponents.all;
```

Remove any other package files with user-defined UNISIM primitives.

Note: If you are using ISE11 for a VHDL design targeting Virtex 2 or Virtex 2 Pro device, you must manually add the unisim_m10i.vhd library before synthesizing the design, because ISE11 does not support these Virtex families. For Verilog designs, the tool automatically add the corresponding unisim_m10i.v file so you do not need to do it manually.

Specifying Xilinx Macros

The synthesis tool provides Xilinx macro libraries that you can use to instantiate components like I/Os, I/O pads, gates, counters, and flip-flops. Using the macros from these libraries allows you to perform a subsequent simulation run without changing your code.

1. To use the Verilog macro library, review the unisim.v macro library in the *installDirectory/lib/xilinx* directory for the available macros. The unisim.v file is automatically added to your project file when you select a Xilinx target device.
2. To use a VHDL library, do the following:
 - Review the unisim.vhd macro library in the *installDirectory/lib/xilinx* directory to check the macros that are available.
 - Add the corresponding library and use clauses to the beginning of the design units that instantiate the macros, as in the following example:

```
library unisim;
use unisim.vcomponents.all;
```

You do not need to add the macro library files to your the source files for your project.

3. Instantiate the macro component in your design.
4. To instantiate an I/O pad with different I/O standards, do the following:
 - Specify the macro library as described in the first two steps.
 - Instantiate the I/O pad component in your design. You can instantiate IBUF, IBUFG, OBUF, OBUFT, and IOBUF components.
 - In the source files, define the generic or parameter values for the I/O standard. Use an IOSTANDARD generic/parameter to specify the I/O standard you want. Refer to the Xilinx documentation for a list of supported IOSTANDARDs. For certain pad types, you can also specify the output slew rate (SLEW) and output drive strength (DRIVE). See *OBUF Instantiation Example*, on page 881 for an example.

OBUF Instantiation Example

The following examples show the declaration of OBUF in macro library files:

VHDL	<pre>component OBUF generic (IOSTANDARD : string := "default"; SLEW : string := "SLOW"; DRIVE : integer := 12); port (O : out std_logic; I : in std_logic;); end component; attribute syn_black_box of OBUF : component is true</pre>
------	---

Verilog	<pre>module OBUF(O, I); /* synthesis syn_black_box */ parameter IOSTANDARD="default"; parameter SLEW="SLOW"; parameter DRIVE=12; output O; input I; endmodule</pre>
---------	---

To use the macro libraries to instantiate I/O pad types, define the generic/parameter values in the Verilog or VHDL source files. The following examples show how to instantiate OBUF pads with an I/O standard value of LVCMOS2, an output slew value of FAST, and an output drive strength of 24.

```
VHDL  Data : OBUF
      generic map (
          IOSTANDARD => "LVCMOS2",
          SLEW => "FAST",
          DRIVE => 24
      )
      port map (
          O => o1,
          I => i1
      );

```

```
Verilog OBUF Data(.O(o1), .I(i1));
defparam Data.IOSTANDARD = "LVCMOS2";
defparam Data.SLEW = "FAST";
defparam Data.DRIVE = 24;
```

The resulting EDIF file contains the following, which corresponds to the instantiations:

```
(instance (
    rename dataZ0 "data")
  (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
  (property iostandard (string "LVCMOS2"))
  (property slew (string "FAST"))
  (property drive (integer 24))
)
```

Specifying Global Sets/Resets and Startup Blocks

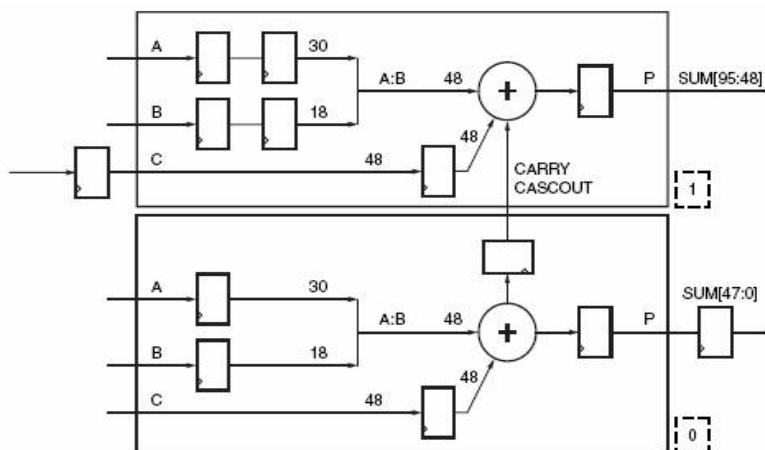
The global set/reset (GSR) resource is a pre-routed signal that goes to the set or reset input of each flip-flop in your design. Using this resource instead of general routing for a set or reset signal can have a significant positive impact on the routability and performance of your design. The synthesis tools infer this resource automatically in most cases, but you can also specify access to the GSR resource with a Xilinx startup block.

1. Specify access to the GSR as follows:
 - To automatically use the GSR if needed, select Implementation Options -> Device, and set Force GSR Usage to auto. With this setting, the tool automatically determines if it needs to use the GSR.
 - To use the GSR, set Force GSR Usage to yes.
 - If you do not want to use the GSR, set Force GSR Usage to no.
2. For Xilinx XC designs, specify global sets/resets (GSR) as follows:
 - For a design with a single GSR, the synthesis tools automatically connect it to a startup block, even if the flip-flops have no sets/resets specified. If you need to change this setting, select Project-> Implementation Options ->Device, and set Force GSR Usage to no. With this setting, all flip-flops must have a set or reset and the set or reset must be the same before GSR is used.
 - For designs with multiple GSRs, the synthesis tool does not automatically create a startup block for GSR. If you still want to use one of the set or reset signals for GSR, you must instantiate a STARTUP_GSR component manually, as described in the next step.
3. To instantiate a start-up block manually, do the following:
 - Go to the *installDirectory/lib/xilinx* directory and locate the appropriate Xilinx startup blocks in either the Verilog (*unisim.v*) or VHDL (*unisim.vhd*) format.
 - Instantiate the startup block component in your design. Where there is more than one component listed, you can use them independently, because the blocks are merged to form a single block in the EDIF file.

Inferring Wide Adders

For Virtex-5 and Virtex-6 designs, you can map wide adder/subtractor structures to DSP48Es. Xilinx architectures let you use cascading DSP48Es and the CARRYCASCOUT pin to support a structure with up to three pipeline registers with different synchronous control signals. It supports two or three signed/unsigned inputs (with carry/borrow).

The following shows the implementation of wide adders with one pipelined register and no pipelined registers as DSP48Es:

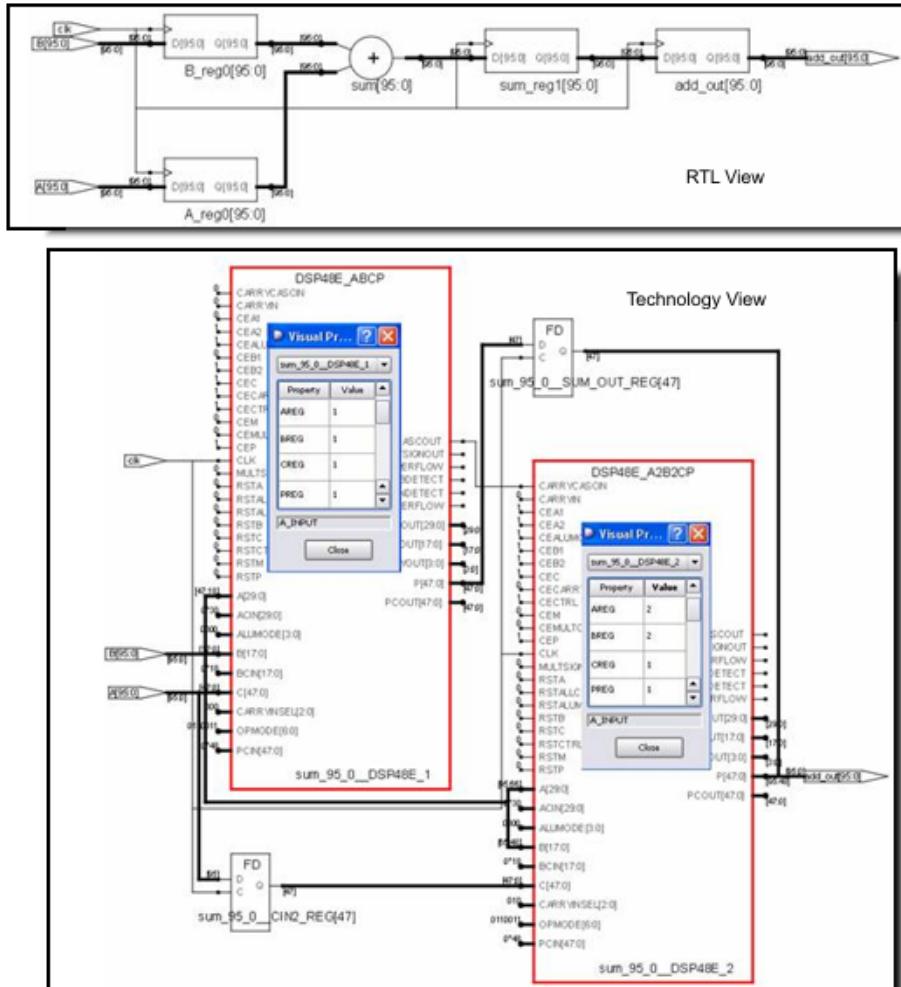


To automatically map to DSP48Es in the synthesis tools, do the following:

1. Make sure the structure you want to map conforms with these rules:
 - The adder/subtractor does not have more than 96 bits.
 - All registers share the same control signals (enables, clocks, reset). Registers with different control signals are mapped to the DSP48E, but they are kept outside the DSP48E.
 - The adder does not have a 48-bit input and a 49-bit output.
 2. Set `syn_dspstyle` to `dsp48`.
- You must set this attribute, or the tool does not infer a DSP48E. See [syn_dspstyle, on page 219](#) for the syntax for this attribute.
3. Synthesize the design.

If your structure has less than three pipelined registers, you see an advisory message in the log file, because three pipelined registers give the best performance.

The following is an example of how the synthesis tool maps an adder->register structure with 96-bit signed input and output to a DSP48E:



Instantiating CoreGen Cores

Predesigned IP cores save on design effort and improve performance. The process for handling IP cores is slightly different for CoreGen and Virtex PCI cores. The following procedure describes how to instantiate a CoreGen module. For Virtex PCI cores, see *Instantiating Virtex PCI Cores*, on page 887.

1. Use the Xilinx CORE generator to create structural EDIF netlists and generate timing and resource usage information for synthesis.
 - For legacy cores, generate a single flat edf netlist file.
 - For newer cores, generate a top-level flat edn or edf netlist file that instantiates ndf files for each hierarchical level in the design.
2. Open the synthesis software, and add the generated files (edf only for legacy cores; edn or edf and ndf for newer cores) to your project.
3. Define the core as a black box by adding the syn_black_box attribute to the module definition line, or by using the Coregen v file. The following is an example of the attribute:

```
module ram64x8(din, addr, we, clk, dout)/* synthesis syn_black_box
*/;
    input [7:0] din;
    input [5:0] addr;
    input we, clk;
    output [7:0] dout;
endmodule;
```

4. Make sure the bus format matches the bus format in the core generator, using the syn_edif_bit_format and syn_edif_scalar_format directives if needed.

```
module ram64x8(din, addr, we, clk, dout)
    /* synthesis syn_black_box syn_edif_bit_format = "%u<%i"
     syn_edif_scalar_format ="%u" */;
```

5. Instantiate the black box in the module or architecture.

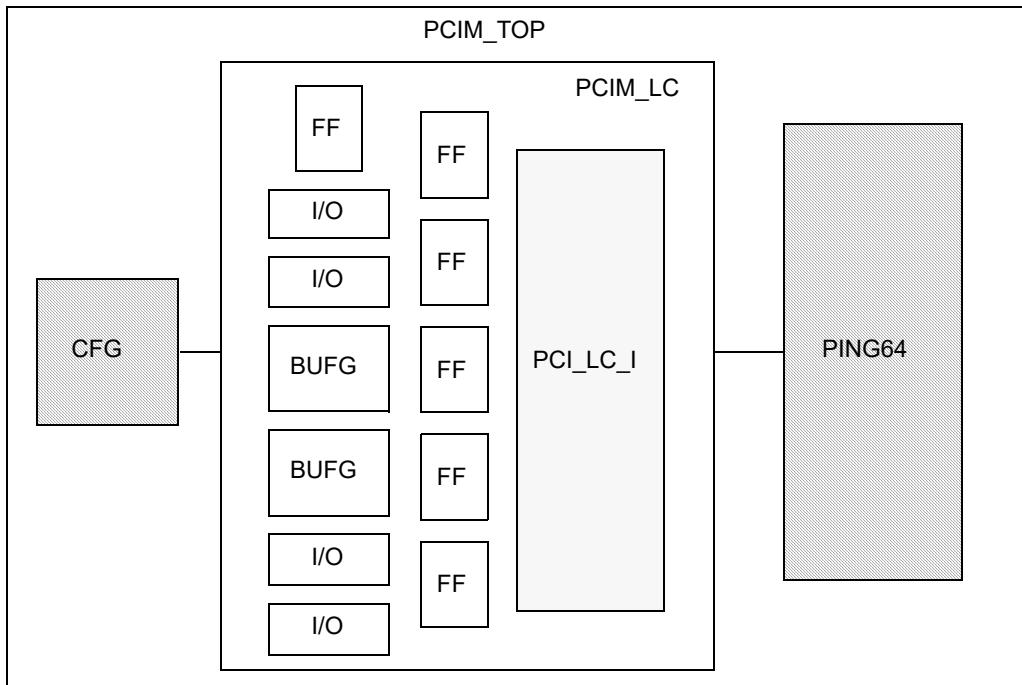
```
ram64x8 r1(din, addr, we, clk, dout);
```

6. Synthesize the design.

If you supplied structural EDIF netlists, the software optimizes the design based on the information in the structural netlists. The generated reports contain the optimization information.

Instantiating Virtex PCI Cores

For Virtex PCI cores, you can use either a top-down or bottom-up methodology. This figure shows a design that is used in the explanations of both methodologies, below.



Bottom-Up Method

The bottom-up method synthesizes lower-level modules first. The synthesized modules are then treated as black boxes and synthesized at the next level. The following procedure refers to the figure shown above.

1. Synthesize the user-defined application (PING64) by itself.
 - Make sure that the Disable I/O Insertion option is on.
 - Specify the `syn_edif_bit_format = "%u<%i>"` and `syn_edif_scalar_format = "%u"` attributes. These attributes ensure that the EDIF bus names

match the Xilinx upper-case, angle bracket style bus names and the Xilinx upper-case net names, respectively.

The software generates an EDIF file for this module.

2. Synthesize the top-level module that contains the PCI core, with the Disable I/O Insertion option enabled and the EDIF naming attributes described in the previous step. Use the following files to synthesize:
 - The top-level module (PCIM_LC) file, with the PCI core (PCI_LC_I) declared as a black box with the `syn_black_box` attribute.
 - A black box file for the core (PCI_LC_I), that only contains information about the PCI core ports. This file is the source file that is generated for simulation, not the `ngo` file.
 - The appropriate synthesis Virtex file (`installDirectory/lib/xilinx`) that contains module definitions of the I/O pads in the top-level module, PCIM_LC.

The software generates an EDIF file for this module.

3. Synthesize the top level (PCIM_TOP) with Disable I/O Insertion off. Use the following files:
 - The source file for CFG.
 - A black box file for PING64.
 - A black box file for PCIM_LC.
 - A top-level file that contains black box declarations for PING64 and PCIM_LC.

The software generates an EDIF file for the top level.

4. Place and route using the Xilinx `ngo` file for the core, and the three EDIF files generated from synthesis: one for each of the modules PING64 and PCIM_LC, and the top-level EDIF file. Select the top-level EDIF file when you run place-and-route.

Top-down Methodology

The top-down method instantiates user application blocks and synthesizes all the source files in one synthesis run. This method can result in a smaller, faster design than with the bottom-up method, because the tool can do cross-boundary optimizations. The following procedure refers to the design shown in the previous figure.

1. Create your own configuration file for your application model (CFG).
2. Edit the top-level source file to do the following:
 - Instantiate your application block (PING64) in the top-level source file.
 - Add the ports from your application.
3. Add the appropriate synthesis Virtex file (*installDirectory/lib/xilinx*) to the project. This file contains module definitions of the I/O pads in the PCIM_LC module.
4. Specify the top-level file in the project.
5. Synthesize your design with the following files:
 - Virtex module definition file (previous step)
 - Source files for top-level design, user application (PING64), PCIM_LC, and CFG
 - Simulation wrapper file for PCI core

The software generates an EDIF file for the top level.

6. Place and route the design using the top-level EDIF file from synthesis and the Xilinx ngo file for the PCI core.

Initializing Xilinx RAM

In addition to the methods described in [Initializing RAMs, on page 528](#), you can also define and forward-annotate Xilinx RAM initialization values as summarized in this table:

Verilog

- \$readmemebh or \$readmemh
See [Initializing RAMs with \\$readmemb and \\$readmemh, on page 532](#).
- INIT property in defparam statement
See [Specifying the INIT Property for Xilinx RAMs \(Verilog\) , on page 890](#).
- INIT property in global comment /* synthesis INIT or /*synthesis INIT xx=<value>
See [Specifying the INIT Property for Xilinx RAMs \(Verilog\) , on page 890](#).

VHDL

- INIT property on label

See [Specifying the INIT Property for Xilinx RAMs \(VHDL\)](#), on page 893.

Attributes

- INIT property in SCOPE

See [Specifying the INIT Property with Attributes](#), on page 894.

- define_attribute statements in the fdc file

See [Specifying the INIT Property with Attributes](#), on page 894.

Note the following differences between the above methods:

- You can use the INIT property with any code. The \$readmemb and \$readmemh system tasks are only applicable in Verilog.
- The Verilog initial values only affect the output of the compiler, not the mapper. They ensure that the synthesis results match the simulation results, and are not forward-annotated.

Specifying the INIT Property for Xilinx RAMs (Verilog)

You can initialize and forward-annotate the values for Xilinx Verilog RAMs by specifying the INIT property. In Verilog, you can do this by using the defparams statement or by specifying the property in a global comment. The following examples illustrate these two methods.

- [Using defparam to Specify Initialization Values for Xilinx RAMs](#), on page 890
- [Using Global Comments to Specify Initialization Values for Xilinx RAMs](#), on page 891

Using defparam to Specify Initialization Values for Xilinx RAMs

- Include defparam statements in the Verilog file, using one statement for each word. Use the following syntax for the INIT property:

defparam name.INIT_xx=value;

name Is the name of the RAM.

xx Indicates the part of the RAM you are initializing. It can be any hex number from 00 to FF.

value Sets the initialization value in hex.

The following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64 x 4 and 256 x 16 = 4K).

```
RAMB4_S4 pkt_len_ram_lo (
    .CLK  (clock),
    .RST  (1'b0),
    .EN   (1'b1),
    .WE   (we),
    .ADDR (address),
    .DI   (data),
    .DO   (q)
);
defparam pkt_len_ram_lo.INIT_00=
"00170016001500140013001200110010000f000e000d000c000b000a00090008 ";
defparam pkt_len_ram_lo.INIT_01=
"00270026002500240023002200210020001f001e001d001c001b001a00190018";
defparam pkt_len_ram_lo.INIT_02=
"00370036003500340033003200310030002f002e002d002c002b002a00290028";
...
defparam pkt_len_ram_lo.INIT_0F=
"0107010601050104010301020101010000ff00fe00fd00fc00fb00fa00f900f8";
```

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx P&R software in the EDIF netlist. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Using Global Comments to Specify Initialization Values for Xilinx RAMs

1. Add the INIT property.
 - Attach the INIT property to the instance as shown:

RAM	/* synthesis INIT = "value" */
Block RAM	/* synthesis INIT_xx = "value" */

- Define the INIT_xx=value property as follows:

xx Indicate the part of the RAM you are initializing with a number from 00 to FF.

value Set the initialization value, in hex. You have 64 hex values in each INIT ($64 \times 4 = 256$ and $256 \times 16 = 4K$), because there are 16 INIT statements.

- Keep the entire statement on one line. Let your editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

2. For RAM, specify a hex value for the INIT statement as shown here:

```
RAM16X1S RAM1 (...) /* synthesis INIT = "0000" */;
```

3. For Virtex block RAM, specify 16 different INIT statements. End the initialization data with a semicolon.

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

The following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64×4 and $256 \times 16 = 4K$).

```
RAMB4_S4 pkt_len_ram_lo (
    .CLK    (clock),
    .RST    (1'b0),
    .EN     (1'b1),
    .WE     (we),
    .ADDR   (address),
    .DI     (data),
    .DO     (q)
)

/* synthesis
INIT_00="00170016001500140013001200110010000f000e000d000c000b000a00090008"
INIT_01="00270026002500240023002200210020001f001e001d001c001b001a00190018"
INIT_02="00370036003500340033003200310030002f002e002d002c002b002a00290028"
...
INIT_OF="0107010601050104010301020101010000ff00fe00fd00fc00fb00fa00f900f8"
*/;
```

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software.

The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Specifying the INIT Property for Xilinx RAMs (VHDL)

1. Add the INIT property.

- Attach the INIT property to the label as shown:

```
RAM      attribute INIT of object : label is "value";
```

```
Block RAM  attribute INIT_xx of object : label is "value";
```

- Keep the entire statement on one line. Let the editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

2. For RAM, specify hex values for the INIT statement as shown:

```
attribute INIT of RAM1 : label is "0000";:
```

3. For Virtex block RAM, specify 16 different INIT statements.

- Define the INIT_xx=value property as follows:

xx Indicate the part of the RAM you are initializing with a number from 00 to FF.

value Set the initialization value, in hex. You have 64 hex values in each INIT ($64 \times 4 = 256$ and $256 \times 16 = 4K$), because there are 16 INIT statements.

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

- End the initialization data with a semicolon.

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Specifying the INIT Property with Attributes

When you set the INIT property in the source code, it is difficult to pass on the values if the RAM instances are mapped to registers. When you specify INIT as an attribute, either in the SCOPE window or the constraint file, you are working with a mapped RAM, and the values are passed to the P&R tool. You can use this method to specify the initialization values for a RAM whether you are using Verilog or VHDL.

1. Compile and map the design.
2. Select the RAM in the SCOPE window.
 - Open SCOPE and go to the Attributes panel.
 - Open the Technology view. Drag and drop the RAM into the window.
3. Define the INIT (RAM) or INIT_xx = *value* (Block RAM) property in SCOPE. Alternatively you can edit the sdc file using define_attribute statements.

xx Indicates the part of the RAM you are initializing with a number from 00 to FF.

value Sets the initialization value, in hex. You have 64 hex values in each INIT ($64 \times 4 = 256$ and $256 \times 16 = 4K$), because there are 16 INIT statements.

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

When you synthesize the design, the software forward-annotates the initialization values as constraints in the sdc file. The following example shows a value of ABBADABAABBADABA defined for INIT_00 and INIT_01 on mem.mem_0_0 in the sdc file:

```
define_attribute {i:mem.mem_0_0} INIT_00 {ABBADABAABBADABA}
define_attribute {i:mem.mem_0_0} INIT_01 {ABBADABAABBADABA}
```

These initialization values are forward-annotated as constraints to the place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Initializing Xilinx Select RAM

For select RAM in some Virtex architectures, you can specify an initial value and have it honored.

1. Specify the initial value in the RTL using \$readmemb or another supported mechanism.

See [Initializing Xilinx RAM, on page 889](#) for details.

If the tool maps the ram1 to a supported select RAM in Virtex 4 and later designs, it automatically distributes the initial value you specified. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Specifying Xilinx Register INIT Values

You can specify initial values for registers so that the RTL, gate-level simulation, and the final implementation results match. You can specify INIT values for registers either with the HDL initialization specification built into Verilog or VHDL, or by adding the synthesis attribute. You can then pass the values to the Xilinx P&R tools.

Both methods are described in the following procedure, but the HDL specification method is recommended.

1. To ensure that the register is not optimized away during synthesis, set the syn_preserve directive on the register in the source code.

Use this directive even if you define the INIT values with a constraint in the sdc file. If you do not have this directive, the register can be removed during optimization.

2. To set a register value using the HDL initialization feature, use the following syntax:

HDL Initialization reg myreg=0;
 initial myreg=0; (Verilog only)

Verilog HDL Initialization

```
reg error_reg = 1'b0;
reg [7:0] address_reg = 8'hff;
```

VHDL HDL Initialization

```
signal tmp: std_logic = '0';
```

This is the preferred method to pass INIT values to the Xilinx place-and-route tools.

- To set a register value using the synthesis attribute, add the attribute to the register in the source code or the constraint file, and specify the INIT value as a string:

Verilog

```
reg [3:0] rst_cntr /* synthesis INIT="1" */;
```

VHDL

```
attribute INIT: string;
attribute INIT of rst_cntr : signal is "1";
```

SDC

```
define_attribute {i:rst_cntr} INIT {"1"}
```

Xilinx ISE 8.2sp3 and later versions require that the INIT value be a string rather than an integer. For code examples, see [INIT Values, on page 802](#) in the *Reference Manual*.

- To specify different INIT values for each register bit on a bus, do the following:
 - Set `syn_preserve` on the register as described in step 1, so that it is not optimized away. You can now either use the HDL specification or set an attribute.
 - To specify the values using the HDL specification, use the syntax as shown in the following examples:

Verilog HDL Bus Initialization

```
reg [7:0] address_reg = 8'hff;
```

VHDL HDL Bus Initialization

```
signal q: std_logic_vector
(11 downto 0) := X"755";
```

- To specify the value with the INIT attribute in the sdc constraint file, set INIT values for the individual register bits on the bus. Specify the register using the i: prefix, with periods as hierarchy separators.

The following specifies INIT values for individual bits of `rst_cntr`, which is part of the `init_attrver` module, under the top-level module:

```
define_attribute {i:init_attrver.rst_cntr[0]} INIT {"0"}
define_attribute {i:init_attrver.rst_cntr[1]} INIT {"1"}
define_attribute {i:init_attrver.rst_cntr[2]} INIT {"0"}
define_attribute {i:init_attrver.rst_cntr[3]} INIT {"1"}
```

5. Synthesize the design.

The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist. Note that the INIT value is forward-annotated as is (as an integer, not binary). You must ensure that the value is specified in the correct format for P&R.

If the register is an asynchronous output register with an initial value, the mapper preserves the initial value and packs the register into the Block RAM.

Packing Registers for Xilinx I/Os

When a register drives an input or output, you might want to pack it in an IOB instead of a CLB, as in these cases:

- The chip interfaces with another, and you have to minimize the register-to-output or input-to-register delay.
- You have limited CLB resources, and packing the registers in an IOB can free up some resources.

To pack registers in an IOB, you set the `syn_useioff` attribute.

1. To globally embed all the registers into IOBs, attach the `syn_useioff` attribute to the module in one of these ways:
 - Add the attribute in the SCOPE window, attaching it to the module, architecture, or the top level. Check the Enable box, set the Attribute column to `syn_useioff`, the Object column to `<global>`, and the attribute value to 1. The constraint file syntax looks like this:

```
define_global_attribute syn_useioff 1
```

- To add the attribute in the Verilog source code, add this syntax to the top level:

```
module global_test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

- To add the attribute in the VHDL source code, add this syntax to the top level architecture declaration:

```
architecture rtl of global_test is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```

For details about attaching attributes using the SCOPE interface and in the source code, see [Specifying Attributes and Directives, on page 137](#).

When set globally, all boundary registers and (OE) registers associated with the data registers are marked with the Xilinx IOB property. This property is forward annotated in the EDIF netlist and used by the Xilinx place-and-route tools to determine how the registers are packed. All marked registers are packed in the corresponding IOBs.

- To apply syn_useioff to individual registers or ports, use one of these methods:

- Add the attribute in the SCOPE window, attaching it to the ports you want to pack, and set the attribute value to 1. The resulting constraint file syntax looks like this:

```
define_attribute {p:q[3:0]} syn_useioff 1
```

- To add the attribute in the Verilog source code, add this syntax:

```
module test is (d, clk, q);
  input [3:0] d;
  input clk;
  output [3:0] q /* synthesis syn_useioff = 1 */;
  reg q;
```

- To add the attribute in the VHDL source code, add syntax as shown inside the entity for the local port:

```
entity test is
  port (d : in std_logic_vector(3 downto 0);
        clk : in std_logic;
        q : out std_logic_vector(3 downto 0));
  attribute syn_useioff : boolean;
  attribute syn_useioff of q : signal is true;
end test;
```

The software attaches the IOB property as described in the previous step, but only to the specified flip-flops. Packing for ports and registers without the attribute is determined by timing preferences. If a register is to be packed into an IOB, the IOB property is attached and forward

annotated. If it is to be packed into a CLB, the IOB property is not forward annotated.

In Virtex designs where the synthesis software duplicates OE registers, setting the `syn_useioff` attribute on a boundary register only enables the associated OE register for packing. The duplicate is not packed, but placed in a CLB. The packed registers are used for data path, and the CLB registers are used for counter implementation.

In Virtex designs where a shift register is at a boundary edge and the `syn_useioff` attribute is enabled, the software extracts only the initial or final SRL16 shift register from the LUT for packing. The shift register that is implemented in the technology view is smaller because of the extraction.

3. If you set multiple `syn_useioff` attributes at different levels of the design, the tool uses the most specific setting (highest priority).

This table summarizes `syn_useioff` priority settings, from the highest priority (register) to the lowest (global):

I/O Type	<code>syn_useioff</code> Value	Description
Register	1	Packs registers into the I/O pad cells, overriding port or global specifications.
	0	Does not pack registers into I/O pad cells, overriding port or global specifications.
Port	1	Packs registers into the I/O pad cells, overriding any global specification.
	0	Does not pack registers into I/O pad cells, overriding any global specification.
Global	1	Packs registers into the I/O pad cells.
	0	Does not pack registers into I/O pad cells.

The `syn_useioff` attribute is supported in the compile point flow.

Inserting Xilinx I/Os and Specifying Pin Locations

By default, the synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). You can change this by enabling Disable IO Insertion in the Device tab of the Implementation Options dialog box. You can also insert I/Os manually by instantiating them.

Whether you use the automatic or manual method, you can specify pin locations for the I/Os with the `xc_loc` attribute. By default, or if no location is specified, the Xilinx tool assigns pin locations automatically.

The following provide details:

- [Assigning Pin Locations for Automatically Inserted Xilinx I/Os](#), on page 900
- [Manually Inserting Xilinx I/Os in Verilog](#), on page 903
- [Manually Inserting Xilinx I/Os in VHDL](#), on page 905

Assigning Pin Locations for Automatically Inserted Xilinx I/Os

The synthesis tool automatically inserts the I/Os (unless you have checked Disable IO Insertion in the Device tab of the Implementation Options dialog box). The following procedure shows you how to assign pin locations for automatically inserted I/Os in a Verilog or VHDL design.

1. Create a new top-level module or entity and instantiate it in your Verilog or VHDL design.

This module/entity holds I/O placement information. Creating this lets you keep your vendor-specific information separate from the rest of your design. Your original design remains technology-independent.

For example, this is a Verilog counter definition:

```
module cnt4 (cout, out, in, ce, load, clk, rst);  
  // Counter definition  
endmodule
```

You create a top-level module that instantiates your design:

```
module cnt4_xilinx (cout, out, in, ce, load, clk, rst);
```

- If you do not want to specify locations, specify the inputs or outputs as usual. The following is an example of Verilog inputs in the top-level module:

```
input ce, load, clk, rst;
```

The Xilinx place-and-route tool automatically places these inputs.

- Optionally, specify I/O locations in the new top-level module, by setting the `xc_loc` attribute.

You can specify the `xc_loc` attribute in the Attribute panel of the SCOPE spreadsheet, as shown below.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
2	<input checked="" type="checkbox"/>	port	in1[2:0]	xc_loc	P2,P3,P	string	Port placement
3	<input checked="" type="checkbox"/>	port	pout1	xc_loc	R1	string	Port placement
4	<input checked="" type="checkbox"/>						

Attributes

Alternatively, you can specify it in the HDL files, as described in [Manually Inserting Xilinx I/Os in Verilog, on page 903](#) and [Manually Inserting Xilinx I/Os in VHDL, on page 905](#). See [xc_loc](#), on page 785 in the *Reference Manual* for syntax details.

The following Verilog code includes `xc_loc` attributes that specify the following locations:

- cout at A1
- out in the top left (TL) of the chip
- in[3] at P20, in[2] at P19, in[1] at P18, and in[0] at P17

```
output cout /* synthesis xc_loc="A1" */;
output [3:0] out /* synthesis xc_loc="TL" */;
input [3:0] in /* synthesis xc_loc="P20,P19,P18,P17" */;
```

- Instantiate the top-level module or entity with the placement information you specified in your design. For example:

```
cnt4 my_counter (.cout(cout), .out(out), .in(in),
    .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule
```

- Synthesize the design.

The synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). The Xilinx place-and-route tool automatically selects locations for I/Os with no `xc_loc` attribute defined. If you specified `xc_loc` settings, they are honored.

VHDL Automatic I/O Insertion Example

```
library synplify;
entity cnt4 is
    port (cout: out bit;
          output: out bit_vector (3 downto 0);
          input: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit );
end cnt4;

architecture behave of cnt4 is
begin
-- Behavioral description of the counter.
end behave;

-- New top level entity, created specifically
-- to place I/Os for Xilinx. This entity is typically
-- in another file, so that your original
-- design stays untouched and technology independent.

entity cnt4_xilinx is
    port (cout: out bit;
          output: out bit_vector (3 downto 0);
          input: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit );

-- Place a single I/O for cout at location A1.
attribute xc_loc : string;
attribute xc_loc of cout: signal is "A1";

-- Place all bits of "output" in the
-- top-left of the chip.
attribute xc_loc of output: signal is "TL";

-- Place input(3) at P20, input(2) at P19,
-- input(1) at P18, and input(0) at P17
attribute xc_loc of input: signal is "P20, P19, P18, P17";

-- Let Xilinx place the rest of the inputs.
end cnt4_xilinx;
```

```
-- New top level architecture instantiates your design.
architecture structural of cnt4_xilinx is
-- Component declaration for your entity.

component cnt4
    port (cout: out bit;
          output: out bit_vector (3 downto 0);
          input: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit );
end component;
begin

-- Instantiate your VHDL design here:
my_counter: cnt4 port map (cout, output, input,
                            ce, load, clk, rst);
end structural;
```

Manually Inserting Xilinx I/Os in Verilog

To insert a Xilinx I/O manually, you must instantiate a black box macro for that I/O from the Xilinx library file. You can then choose to assign it a location or have the Xilinx tool automatically select one for it.

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the *installDirectory/lib/xilinx/unisim.v* macro library file to the *top* of the source files list for your project.
2. Create instances of I/Os by instantiating a black box in your Verilog source code.

These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To insert an I/O manually and specify pin locations, do the following:

1. Create a new top-level module and instantiate your Verilog design.
2. Add the *installDirectory/lib/xilinx/unisim.v* macro library file to the *top* of the source files list for your project.
3. Create instances of I/Os by instantiating a black box in your Verilog source code.

4. Specify I/O locations by adding the `xc_loc` attribute to the I/Os.

See [Verilog Manual I/O Insertion Example, on page 904](#) for an example of the code. The Xilinx tool honors any locations assigned with the `xc_loc` attribute, and automatically selects locations for any remaining I/Os without definitions.

Verilog Manual I/O Insertion Example

```
module cnt4 (cout, out, in, ce, load, clk, rst);
/* Your counter definition goes here, */
endmodule
/* Create a top level to place I/Os specifically
   for Xilinx. Any top level pins which do not have
   I/Os will be automatically inserted */
module cnt4_xilinx(cout, out, in, ce, load, clk, rst);
output [3:0] out;
output cout;
input [3:0] in;
input ce, load, clk, rst;wire [3:0] out_c, in_c;
wire cout_c;

/* The xc_loc attribute can be added right after the
   instance name like that shown below, or right before
   the semicolon. */

IBUF i3 /* synthesis xc_loc="P20" */ (.O(in_c[3]), .I(in[3]));
IBUF i2 /* synthesis xc_loc="P19" */ (.O(in_c[2]), .I(in[2]));
IBUF i1 /* synthesis xc_loc="P18" */ (.O(in_c[1]), .I(in[1]));
IBUF i0 /* synthesis xc_loc="P17" */ (.O(in_c[0]), .I(in[0]));

OBUF o3 /* synthesis xc_loc="TL" */ (.O(out[3]), .I(out_c[3]));
OBUF o2 /* synthesis xc_loc="TL" */ (.O(out[2]), .I(out_c[2]));
OBUF o1 /* synthesis xc_loc="TL" */ (.O(out[1]), .I(out_c[1]));
OBUF o0 /* synthesis xc_loc="TL" */ (.O(out[0]), .I(out_c[0]));

OBUF cout_p /* synthesis xc_loc="BL" */ (.O(cout), .I(cout_c));

cnt4 it(.cout(cout_c), .out(out_c), .in(in_c),
       .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule
```

Manually Inserting Xilinx I/Os in VHDL

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the corresponding library and use clauses to the beginning of your design units that instantiate the macros.

```
library unisim;
use unisim.vcomponents.all;
```

The Xilinx unisim.vhd macro library is always visible in the synthesis tool, so do not add this library file to the source files list for your project. To see which design units are available, use a text editor to view the file located in the *installDirectory/lib/xilinx* directory. Do not edit this file in any way.

2. Create instances of I/Os by instantiating a black box in your Verilog source code.

These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To insert an I/O manually and specify pin locations, do the following:

1. Create a new top-level module and instantiate your VHDL design.
2. Instantiate the Xilinx I/Os.
3. Add the appropriate library and use clauses to the beginning of design units that instantiate the I/Os.

```
library unisim;
use unisim.vcomponents.all;
```

See the source code in [VHDL Manual I/O Insertion Example](#), on page 906 for an example.

4. To specify I/O locations, add the `xc_loc` attribute to the I/O instances for which you want to specify the locations.
5. If you leave out the `xc_loc` attribute, the Xilinx place-and-route tool will choose the locations.

VHDL Manual I/O Insertion Example

The following example is a behavioral D flip-flop with instantiated data input I/O. The other ports will have synthesized I/Os.

```
library ieee, synplify;
use synplify.attributes.all;
use ieee.std_logic_1164.all;
-- Library and use clauses for access to the Xilinx Macro Library.
library unisim;
use unisim.vcomponents.all;

entity place_example is
    port (q: out std_logic;
          d, clk: in std_logic );
end place_example;

architecture behave of place_example is
    signal dz: std_logic;

attribute xc_loc of I1: label is "P3";
begin
I1: IBUF port map (I=>d,O=>dz);

process (clk) begin
    if rising_edge(clk) then
        q<=dz;
    end if;
end process;
end behave;
```

Working with Xilinx Buffers

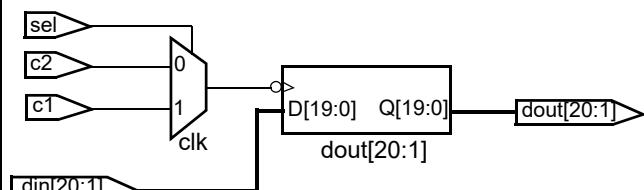
By default, the synthesis tools do not automatically infer Xilinx buffers. If you want the tools to infer Xilinx buffers, you must use attributes, as described below.

1. To infer BUFGMUX components, do the following:
 - Attach the `syn_insert_buffer` attribute to the mux instance. If you need information on how to do this, see [Specifying Attributes and Directives, on page 137](#).
 - Set the attribute value to `bufgmx`. When you set this value, the tool infers a `BUFGMUX_1` if the muxed clock operates on the negative edge;

otherwise it infers a BUFGMUX. If you do not specify this value, by default the tool infers the LUT that drives the BUFG.

```
module
bufgmxux_1(c1,c2,sel,din,d
out);
input c1,c2,sel;
input [20:1] din;
output reg [20 : 1] dout;
wire clk;

assign clk = sel ? c1 :
```



For details about the `syn_insert_buffer` syntax, see [syn_insert_buffer, on page 339](#) in the *Attribute Reference Manual*

2. To infer IBUFDS, IBUFGDS, OBUFDS, OBUFTDS, and IOBUFDS differential buffers, do the following:
 - Attach the `syn_diff_io` attribute to the inputs of the buffer.
 - Set the value to 1 or true.

For details about the `syn_diff_io` syntax, see [syn_diff_io, on page 183](#) in the *Attribute Reference Manual*.

The `syn_diff_io` attribute is supported in the compile point flow.

Working with Xilinx Regional Clock Buffers

The Xilinx regional clock buffer (BUFR) available for Virtex-6, Virtex-5, and Virtex-4 devices is a special buffer used to connect the clock nets in the same region and adjacent regions, independent of the global clock tree. The BUFR can drive the I/O logic and logic resources (CLB, block RAM, and DSP) in the existing and adjacent clock regions. They can be driven by clock capable pins, local interconnects, Gigabit Transceiver (GT), and Mixed-Mode Clock Manager (MMCM) high-performance clocks. The BUFR can be used as a clock divider in low power and low skew operations.

By default, the synthesis tools do not automatically infer the Xilinx BUFR. If you want the tools to infer Xilinx regional clock buffers, you must use the attribute, as described below.

1. To infer BUFR primitives, do the following:

- Apply the `syn_insert_buffer` attribute on a port or net.
- Set the attribute value to BUFR.

For details about the `syn_insert_buffer` syntax, see [syn_insert_buffer, on page 339](#) in the *Attribute Reference Manual*

2. Check the log file for the number of BUFRs connected in the region. The report specifies:

- The output of the BUFR used on the clock net should be defined as the derived clock and the timing report should be specified for the clock signal.
- If inferred

Clock Buffers:

Inserting Clock buffer on net `clk1_en`, Inserting Clock buffer for port `clk`,

- Not inferred

Warning: BUFR not inserted on net <name>

- Resource utilization

Mapping to part: `xc6vlx75tff484-1`

Cell usage:

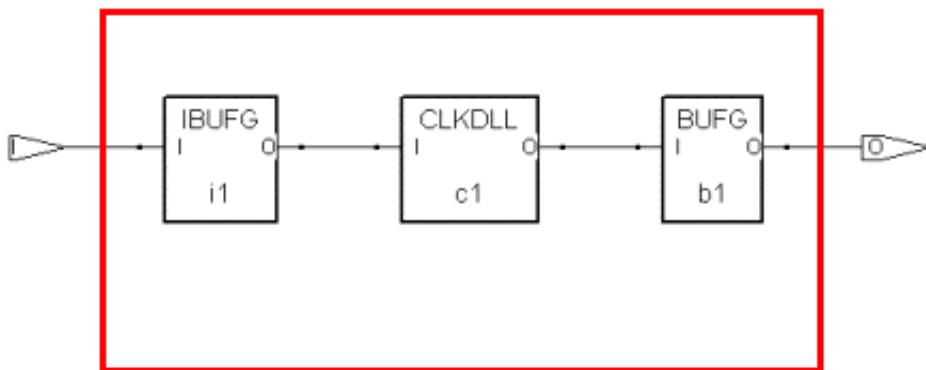
BUFR 2 uses

Using Clock Buffers in Virtex Designs

The software can infer a buffer called BUFGDLL that includes the CLKDLL primitive. BUFGDLL consists of an IBUFG followed by a CLKDLL (Clock Delay Locked Loop) followed by a BUFG. To use this CLKDLL primitive, you must specify the `xc_clockbuftype` attribute. The following steps show you how to add the attribute in SCOPE or the HDL files.

1. To specify the attribute in the SCOPE window, use the procedure described in [Specifying Attributes Using the SCOPE Editor, on page 143](#) to add the `xc_clockbuftype` attribute to a port.

The software infers a buffer as shown in the following figure.



The output EDIF netlist contains text like the following:

```
(instance clk_ibuf (viewRef PRIM (cellRef BUFGDLL (libraryRef VIRTEX) ) )
```

2. To specify the attribute in Verilog, add the attribute as shown in this example.

```
module test(d, clk, rst, q);
    input [1:0] d;
    input clk /* synthesis xc_clockbuftype = "BUFGDLL" */, rst;
    output [1:0] q;
    //other coding
```

3. To specify the attribute in VHDL, add the attribute as shown in this example.

```
entity test_clkbuftype is
    port (d: in std_logic_vector(3 downto 0);
          clk, rst : in std_logic;
          q : out std_logic_vector(3 downto 0)
        );
    attribute xc_clockbuftype of clk : signal is "BUFGDLL";
end test_clkbuftype
```

Working with Clock Skews in Xilinx Virtex-5 Designs

The Synplify Premier software supports clock skews for Virtex-5 devices and the SRM clock insertion delay property (Clock input arrival_time). Clock insertion delay models are included for the following components:

- DCM
- BUFG
- BUFR
- BUFIO
- Flip-flops generating clocks

This feature ensures that cross-clock paths are compared correctly. Also, it has a large impact on timing constraints for I/O paths, since any clock delay will be added to the output delay and subtracted from the setup delay. This results in improved timing correlation between the Synplify Premier software and Xilinx timing.

Example1: Calculating Slack with Clock Skew

Clock skew is utilized to calculate the slack in the following Virtex-5 example:

- Source DCM (clock insertion delay = 0.000ns)

$$\begin{aligned} \text{Requested Period: } & 5.000 \\ - (\text{Setup Time}): & 0.004 \\ + (\text{Clock Delay at Ending Point}): & 4.157 \\ + (\text{Clock Latency at Ending Point}): & 0.000 \\ = \text{Required Time: } & 9.153 \\ - (\text{Propagation Time}): & 0.746 \\ - (\text{Clock Latency at Starting Point}): & 0.000 \\ = \text{Slack (non-critical)}: & 8.407 \end{aligned}$$

Example 2: Calculating Slack Using Clock Skew

Clock skew is utilized to calculate the slack in the following Virtex-5 example:

- Source IBUFG (clock insertion delay = 4.157ns)
- Load DCM (clock insertion delay = 0.000ns)

```

Requested Period: 5.000
  - (Setup Time): 0.004
+ (Clock Latency at Ending Point): 0.000

= Required Time: 4.996
  - (Propagation Time): 0.745
- (Clock Delay at Starting Point): 4.157
- (Clock Latency at Starting Point): 0.000
= Slack (critical): 0.094

```

The Synplify Premier software does not automatically forward annotate constraints for derived clocks. Therefore, a clock generated from a set of flip-flops and logic requires you to add a constraint in the UCF file. As a recommendation, derive the clock period the same as the original clock and add a 2-cycle multicycle path from the clock to itself. Better solutions will be provided in the future.

Instantiating Special I/O Standard Buffers for Virtex

The software supports all the I/O Virtex standards, like HSTL_*, CTT, AGP, PC133_*, PC166_*, etc. You can either instantiate these primitives directly, or specify them with the xc_padtype attribute.

1. To instantiate I/O buffers, use code like the following to specify them.

```

module inst_padtype(a, b, clk, rst, en, bidir, q) ;
  input [0:0] a, b;
  input clk, rst, en;
  inout bidir;
  output [0:0] q;

  reg [0:0] q_int;
  wire a_in, q_in;
  IBUF_AGP i1 (.O(a_in), .I(a));
  IOBUF_CTT i2 (.O(q_in), .IO(bidir), .I(q_int), .T(en));
  OBUF_F_12 o1 (.O(q), .I(q_in));

```

```

always @ (posedge clk or posedge rst)
  if (rst)
    q_int <= 1'b0;
  else
    q_int <= a_in & b;
endmodule

```

2. To specify the I/O buffers with an attribute, add the attribute in the SCOPE window (refer to [Specifying SCOPE Constraints, on page 210](#) for details) or in the source code, as the following example illustrates.

```

module inst_padtype(a, b, clk, rst, en, bidir, q) ;
  input [0:0] a /* synthesis xc_padtype = "IBUF_AGP" */;
  input clk, rst, en;
  inout bidir /* synthesis xc_padtype = "IOBUF_CTT" */;
  output [0:0] q /* synthesis xc_padtype = "OBUF_F_12" */;

  reg [0:0] q_int;

  assign q = bidir;
  assign bidir = en ? q_int : 1'bz;
  always @ (posedge clk or posedge rst)
    if (rst)
      q_int <= 1'b0;
    else
      q_int <= a & b;
endmodule

```

Specifying RLOCs

RLOCs are relative location constraints. They let you control placement in critical sections, thus improving performance. You specify RLOCs using three attributes, `xc_map`, `xc_rloc`, and `xc_uset`. As with other attributes, you can define them in the source code, or in the SCOPE window.

You can also specify RLOCs directly, as described in [Specifying RLOCs and RLOC_ORIGINS with the synthesis Attribute, on page 914](#).

1. Create the modules you want to constrain, and specify the kind of Xilinx primitive you want to map them to, using the `xc_map` attribute. The modules can have only one output.

Family	xc_map Value	Max. Module Inputs
Spartan families	fmap	4
	hmap	3
Virtex and Spartan-3 families	lut	4

This Verilog example shows a 4-input Spartan XOR module:

```
module fmap_xor4(z, a, b, c, d) /* synthesis xc_map=fmap */ ;
output z;
input a, b, c, d;
assign z = a ^ b ^c ^d;
endmodule
```

This is the equivalent VHDL example:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fmap_xor4 is
    port (a: in std_logic;
          b: in std_logic;
          c: in std_logic;
          d: in std_logic
        );
end fmap_xor4;

architecture rtl offmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl: architecture is "fmap";
begin
    z <= a xor b xor c xor d;
end rtl;
```

2. Instantiate the modules you created at a higher hierarchy level.
3. Group the instances together (xc_uset attribute) and specify the relative locations of instances in the group with the xc_loc attribute.

This example shows the Verilog code for the top-level CLB that includes the 4-input module in the previous example:

```
module clb_xor9(z, a);
output z;
input [8:0] a;
wire x03, x47;
fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
(z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
(z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
(z, z03, z47, a[8]);
//Code for Virtex differs because it includes the slice
fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
(z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
(z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.S1" */
(z, z03, z47, a[8]);endmodule
```

4. Create a top-level design and instantiate your design.

Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute

You can specify RLOCs and RLOC_ORIGINs with the synthesis attribute, and then pass them to the Xilinx P&R tools. Alternatively, you can specify RLOCs using the three attributes described in [Specifying RLOCs, on page 912](#).

1. In the source code, use the synthesis attribute to specify the RLOC and RLOC_ORIGIN values:

<pre>Verilog /* synthesis RLOC_ORIGIN="X0Y2" RLOC="X0Y0" */;</pre>	<pre>VHDL attribute RLOC_ORIGIN : string; attribute RLOC_ORIGIN of behave : architecture is "X0Y2"; attribute RLOC : string; attribute RLOC of q : signal is "X0Y0";</pre>
--	--

For code examples, see [RLOC Constraints, on page 803](#) in the *Reference Manual*.

2. To specify different RLOC and RLOC_ORIGIN values for bits on a bus, do the following:

- Specify the RLOC_ORIGIN for the Verilog module or VHDL architecture in the source file. See step 1 for the syntax.
- Define RLOCs for the individual register bits as constraints in the `sdc` file. Do not define RLOCs for individual bits in the source code, or you will get a Xilinx ISE error.

```
define_attribute {i:tmp[0]} RLOC {"X3Y0"}  
define_attribute {i:tmp[1]} RLOC {"X2Y0"}  
define_attribute {i:tmp[2]} RLOC {"X1Y0"}  
define_attribute {i:tmp[3]} RLOC {"X0Y0"}
```

3. Synthesize the design.

The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist.

Xilinx Optimizations

The synthesis tools offer various optimizations for Xilinx designs. The following describe the optimizations in more detail:

- [Fanout Limits in Xilinx Designs](#), on page 916
- [Pipelining in Xilinx Designs](#), on page 918
- [Retiming in Xilinx Designs](#), on page 918
- [Gated and Generated Clocks in Xilinx Designs](#), on page 919
- [Sequential Optimizations in Xilinx Designs](#), on page 919
- [Compile Point Remapping in Xilinx Designs](#), on page 920
- [Clock Skew Estimation for Xilinx Designs](#), on page 921
- [Global Set/Resets and Startup Blocks](#), on page 921
- [Advanced LUT Combining](#), on page 922

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 917), and the `syn_maxfan` attribute ([The syn_maxfan Attribute for Xilinx Designs](#), on page 917).

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 579 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 942).

For tips on setting and using fanout limits, see [Setting Fanout Limits, on page 577](#) and [Controlling Buffering and Replication, on page 579](#) 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 375](#) 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

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

Synplify Pro, Synplify Premier

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.

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 465](#) for the syntax.

See [Pipelining, on page 559](#) in the *User Guide* for a step-by-step procedure on using pipelining.

Retiming in Xilinx Designs

Synplify Pro, Synplify Premier

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.

Set global retiming either from the Project view check box or from the Device tab under the Impl Options button. Use the `syn_allow_retimimg` attribute to enable or disable retiming for individual flip-flops. See [syn_allow_retimimg, on page 99](#) for syntax details, or [Retiming, on page 563](#) in the *User Guide* for a procedure.

Gated and Generated Clocks in Xilinx Designs

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 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 828](#) 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 881](#) in the *User Guide*.

Sequential Optimizations in Xilinx Designs

The Disable Sequential Optimization option determines when sequential optimizations are performed. Deselect the checkbox or setting the option to 0 (the defaults) enables sequential optimizations to be performed. The unused registers are always removed from the design regardless of this option setting. Disabling sequential optimizations (checking the Disable Sequential Optimizations checkbox) can increase delay times and area requirements, which effectively disables the FSM Compiler and FSM Explorer. The Tcl equivalent for this device option is `set_option -no_sequential_opt 1 | 0`.

Compile Point Remapping in Xilinx Designs

Synplify Pro, Synplify Premier

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 626](#) and [Compile Point Basics, on page 606](#) for information about the flow and compile points. Compile points are supported for all Virtex technologies and Spartan-II or later technologies.

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 <i>not</i> 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 <i>interface</i> 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 <i>updated</i> 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: <ul style="list-style-type: none">• The compile point is remapped, and the Update Compile Point Timing Data option is enabled.• The interface of the compile point is changed.

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

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 882](#) 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. |
-

Advanced LUT Combining

Virtex-5 and Spartan-6 or Newer Technologies

This option is supported in Xilinx Vivado and ISE 10.1 (sp1) or later P&R versions. The Enable Advanced LUT Combining (Implementation Options->Device) option is used to prepare the netlist for advanced LUT combining, so that the Xilinx P&R tool can pack into the LUT6_2 depending on available resources. You can also enable this mode using the equivalent Tcl command:

```
set_option -enable_prepacking 1.
```

Reporting the Number of LUTS

After synthesizing a design with the Enable Advanced LUT Combining option, the total number of LUTS reported in the SRR and the AREASRR files may not match. Here is why they can differ.

- The SRR file reports the total LUT usage in the section called Distribution of All Consumed Luts as follows:

RAM LUTS (RAM implemented as distributed RAM or LUTRAM) + Actual LUTS (Combinational logic) - Shared LUTS (Shared LUT6_2)

For the example shown below, the total number of LUTS reported in the SRR file is 213 (8 + 227 - 44/2).

```

Resource Usage Report for eight_bit_uc

Mapping to part: xc6vlx240tff1759-1
Cell usage:
RAM32M      2 uses
FD          5 uses
FDC         85 uses
FDCE        56 uses
FDPE        24 uses
GND          7 uses
MUXCY       2 uses
MUXCY_L     25 uses
MUXF7       20 uses
VCC          7 uses
XORCY       26 uses
LUT1          1 use
LUT2         31 uses
LUT3         12 uses
LUT4         34 uses
LUT5         43 uses
LUT6        106 uses

I/O ports: 26
I/O primitives: 26
IBUF          1 use
IBUFG         1 use
IOBUF        24 uses

BUFG          1 use

I/O Register bits:          0
Register bits not including I/Os: 170 (0%)

RAM/ROM usage summary
Simple Dual Port Rams (RAM32M): 2

Global Clock Buffers: 1 of 32 (3%)

Total load per clock:
  eight_bit_uc|clock: 172

Mapping Summary:
Total LUTs: 213 (0%)

```

Distribution of All Consumed LUTs = RAM + LUT1 + LUT2 + LUT3 + LUT4 + LUT5 + LUT6 - HLUTNM/2
 Distribution of All Consumed Luts 213 = 8 + 1 + 31 + 12 + 34 + 43 + 106 - 44/2

- However, the total LUT count in the AREASRR is calculated differently than the SRR file as follows:

SRR LUT Count (Combinational logic) - RAM LUTS (RAM implemented as distributed RAM or LUTRAM) + Shared LUTS (Shared LUT6_2)

In the following example, the total number of LUTS (Combinational Logic) reported in the AREASRR file is 227 (213 - 8 + 44/2) as shown below.

Part: XC6VLX240TFF1759-1 (Xilinx)

Utilization report for Top level view: eight_bit_uc

SEQUENTIAL ELEMENTS

Name	Total elements	Total area	Utilization	Notes
REGISTERS	170	170	100 %	

Total SEQUENTIAL ELEMENTS in the block eight_bit_uc: 170 (33.01 % Utilization)

COMBINATIONAL LOGIC

Name	Total elements	Total area	Utilization	Notes
LUTS	227	227	100 %	
Shared LUTS	44	4294967274	100 %	
MUXCY	27	27	100 %	
KORCY	26	26	100 %	

Total COMBINATIONAL LOGIC in the block eight_bit_uc: 324 (62.91 % Utilization)

Distributed Memory

Name	Total elements	Total area	Utilization	Notes
Distributed RAM	2	8	100 %	

Total Distributed Memory in the block eight_bit_uc: 2 (0.39 % Utilization)

Note that the total combinational logic includes shared LUTS and MUXCY and XORCY combinational logic as well, which totals to 324. Distributed RAM count is listed separately in the hierarchical area report.

Xilinx Device Mapping Options

To achieve optimal design results, set the correct implementation options. This section contains options that appear on the Device and Options tab of the Implementation Options panel for Xilinx devices. These options control the following:

- target technology and related parameters
- mapping options to use during synthesis
- output and output formats
- results directories

See Also

- [Virtex and Spartan-II and Later Technologies](#), on page 925
- [Xilinx CPLDs](#), on page 935

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 925
- [Virtex and Spartan-II and Later set_option Tcl Command](#), on page 928
- [Virtex and Spartan-II and Later project Tcl Command](#), on page 935

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 447 and [Constraints Panel](#), on page 450.

The following table lists device mapping options (Device tab) for the Virtex and Spartan-II and later 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 475 in the <i>User Guide</i> for more information.
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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 867 for details.
Disable Sequential Optimizations	Enables or disables sequential optimizations for the design. See Sequential Optimizations in Xilinx Designs , on page 919 .
Distributed Synthesis	<i>Synplify Premier</i> When enabled, distributed synthesis automatically splits large designs into smaller sub-designs for parallel processing on separate machines (or on the same machine) to improve runtime. The tool uses the Synopsys CDPL (Common Distributed Processing Library) scheme for distributed processing. See Using Distributed Processing, on page 808 .
Enable Advanced LUT Combining	Prepares the netlist for advanced LUT combining in Virtex-5 and Spartan-6 or newer designs, so that Xilinx Vivado and ISE P&R can pack into the LUT6_2 depending on available resources. The default is on. See Advanced LUT Combining , on page 922 for additional details.
Exploratory Place and Route	<i>Synplify Premier</i> Runs different place-and-route configurations for the design that can favorably affect routability of the design.

Fanout Guide	<i>Syplify Premier</i> 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 916 for background information. For tips on setting and using fanout limits, see Setting Fanout Limits, on page 577 and Controlling Buffering and Replication, on page 579 in the <i>User Guide</i> .
Implicit Initial value support for Instantiated primitives	<i>Synplify Premier</i> When enabled, automatically loads implicit initial values for the technology primitives used during optimization (the advanced synthesis strategy must also be enabled). See Using Implicit Initial Values, on page 536 in the <i>User Guide</i> .
Maximum Parallel Jobs for Exploratory Place and Route	<i>Syplify Premier</i> Specifies the number of CPUs available to use for parallel processing. Each parallel job uses one CPU. The default is 4.
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.

Synthesis Strategy	<i>Synplify Premier</i> Specifies a synthesis strategy to use for the design. Synthesis modes include: <ul style="list-style-type: none">• advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but with longer runtimes compared to other modes. This is the default.• base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes.• fast – Reduces the number of optimizations performed to improve the synthesis runtime.• routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.
Update Compile Point Timing Data	<i>Synplify Pro, Synplify Premier</i> 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 920, for details.
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 1142 in the <i>User Guide</i> .

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 928 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 149](#) 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 933 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 444).
-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 444).
-package packageName	Specifies the package for the implementation. Refer to the Implementation Options dialog box for available choices (Implementation Options Command, on page 444).
Optimization Options	
-automatic_compile_point 1 0	Enables or 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 627 in the <i>User Guide</i> .
-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 867 for a description.

OPTION	DESCRIPTION
-distributed_synthesis 1 0	<i>Synplify Premier</i> When enabled, distributed synthesis automatically splits large designs into smaller sub-designs for parallel processing on separate machines (or on the same machine) to improve runtime. The tool uses the Synopsys CDPL (Common Distributed Processing Library) scheme for distributed processing. See Using Distributed Processing, on page 808 .
-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 922 for additional details.
-fanout_limit 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 916 for details.
-fix_gated_and_generated_clocks 1 0	Performs gated- and generated-clock optimizations when enabled. See Working with Gated Clocks, on page 828 and Optimizing Generated Clocks, on page 881 in the <i>User Guide</i> for details.
-hamming3 1 0	<i>Synplify Premier</i> 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.
-max_parallel_par_explorer value	<i>Syplify Premier</i> Specifies the number of CPUs available to use for parallel processing. Each parallel job uses one CPU. The default is 4.
-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 <code>false</code> or 0; therefore, sequential optimizations are performed. Values can be 1 or <code>true</code> , 0 or <code>false</code> . Disabling sequential optimizations can increase delay times and area requirements, and effectively disables the FSM Compiler and FSM Explorer.

OPTION	DESCRIPTION
-par_explorer 1 0	<i>Synplify Premier</i> Runs different place-and-route configurations for the design that can favorably affect routability of the design.
-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 947 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.
-pipe 1 0	<i>Synplify Pro, Synplify Premier</i> 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 559 of the <i>User Guide</i> 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 in the compiler. See Sharing Resources, on page 581 in the <i>User Guide</i> for information about using it.
-retiming 1 0	<i>Synplify Pro, Synplify Premier</i> 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 563 of the <i>User Guide</i> for information on how to use it.
-run_prop_extract 1 0	Determines whether the tool extracts properties for annotation. See Annotating Timing Information in the Schematic Views, on page 475 in the <i>User Guide</i> for more information.

OPTION	DESCRIPTION
-rw_check_on_ram 1 0	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
-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.
-support_implicit_init_netlist 1 0	<i>Synplify Premier</i> When enabled, implicit initial values for the technology primitives are automatically loaded and used during optimization (the <code>-synthesis_strategy</code> advanced option must also be enabled). See Using Implicit Initial Values, on page 536 in the <i>User Guide</i> .
-symbolic_fsm_compiler 1 0	Enables or 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 585 in the <i>User Guide</i> for details of its use.
-autosm 1 0	

OPTION	DESCRIPTION
-synthesis_strategy advanced base fast routability	<p><i>Synplify Premier</i></p> <p>Specifies the synthesis strategy to use for the design. Synthesis modes include:</p> <ul style="list-style-type: none"> • advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but may require longer runtimes compared to other modes. This is the default. • base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes. • fast – Reduces the number of optimizations performed to improve the synthesis runtime. • routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.
-update_models_cp 1 0	<p><i>Synplify Pro, Synplify Premier</i></p> <p>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 920, for details.</p> <ul style="list-style-type: none"> • 1 Forces remapping of parents • 0 Does not remap parents
-use_fsm_explorer 1 0	<p><i>Synplify Pro, Synplify Premier</i></p> <p>Enables or disables the FSM Explorer. See Running the FSM Explorer, on page 589 in the <i>User Guide</i> for information about using it.</p>

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	QPro Virtex Hi-Rel	qprovirtexh
Spartan-IIIE	spartan2e	QPro Virtex Rad-Hard	qprovirtexr
Spartan-3	spartan3	QPro Virtex-II Military	qprovirtex2
Spartan-3E	spartan3e	QPro Virtex-II Rad Tolerant	qprorvirtex2

Technology	Value	Technology	Value
Spartan-IIIE Automotive	asparten2e	QPro Virtex-E Military	qprovirtex-em
Spartan-3 Automotive	asparten3	QPro Virtex Hi-Rel	qprovirtexh
Spartan-6	spartan6	Defense-Grade Artix7	Defense-Grade-Artix7
Virtex-II	virtex2	Defense-Grade Kintex7	Defense-Grade-Kintex7
Virtex-II Pro	virtex2p	Defense-Grade Spartan-6Q	QProSpartan6
Virtex-4	virtex4	Defense_Grade Virtex-4Q	QProVirtex4
Virtex-5	virtex5	Defense-Grade Virtex-5Q	QProVirtex5
Virtex-6	virtex6	Defense-Grade Virtex-6Q	QProVirtex6
Virtex-7	virtex7	Defense-Grade Virtex7	Defense-Grade-Virtex7
Kintex-7	kintex7	Defense-Grade Zynq	Defense-Grade-Zynq
Artix-7	artix7	Space-Grade Virtex-4QV	QProRVirtex4
Kintex UltraScale FPGAs	Kintex-UltraScale-FPGAs	Zynq	zynq
Virtex UltraScale FPGAs	Virtex-UltraScale-FPGAs	Zynq Automotive	Azynq
Kintex UltraScale+ FPGAs	Kintex-UltraScalePlus-FPGAs	Zynq UltraScale+ FPGAs	Zynq-UltraScalePlus-FPGAs
Virtex UltraScale+ FPGAs	Virtex-UltraScalePlus-FPGAs		

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 111 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 <code>edif</code> (lower-case letters).

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 935
- [Xilinx CPLD set_option Tcl Command](#), on page 937
- [Xilinx CPLD Technologies](#), on page 939
- , on page 878

The applicable CPLD technologies include CoolRunner-II, CoolRunner-II Automotive and CoolRunner XPLA3.

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 447](#) and [Constraints Panel, on page 450](#).

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 475 in the <i>User Guide</i> for more information.
Automatic Read/Write Check Insertion for RAM	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 509 .
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 867 for details.
Disable Sequential Optimizations	Enables or disables sequential optimizations for the design. See Sequential Optimizations in Xilinx Designs , on page 919 .
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 916 for background information. For tips on setting and using fanout limits, see Setting Fanout Limits, on page 577 and Controlling Buffering and Replication, on page 579 in the <i>User Guide</i> .
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.

Synthesis Strategy*Synplify Premier*

Specifies a synthesis strategy to use for the design. Synthesis modes include:

- advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but with longer runtimes compared to other modes. This is the default.
- base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes.
- fast – Reduces the number of optimizations performed to improve the synthesis runtime.
- routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.

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 939 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 444 .
-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 444 .

OPTION	DESCRIPTION
-package packageName	Specifies the package for the implementation. Refer to the Implementation Options dialog box for available choices - see Implementation Options Command, on page 444 .
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 and FSM Explorer are effectively disabled. See Sequential Optimizations in Xilinx Designs , on page 919 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.
-resource_sharing 1 0	Enables/disables resource sharing. See Sharing Resources, on page 581 in the <i>User Guide</i> for information about using it.
-rw_check_on_ram 1 0	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the syn_ramstyle attribute, see syn_ramstyle, on page 509 .
-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 585 in the <i>User Guide</i> for details of its use.
-autosm 1 0	

OPTION	DESCRIPTION
-synthesis_strategy advanced base fast routability	<p><i>Synplify Premier</i></p> <p>Specifies a synthesis strategy to use for the design. Synthesis modes include:</p> <ul style="list-style-type: none"> • advanced – Performs additional optimizations during logic synthesis to provide an output netlist with better QoR, but may require longer runtimes compared to other modes. This is the default. • base – Generates Synplify Pro results. This is the same as turning off advanced and fast modes. • fast – Reduces the number of optimizations performed to improve the synthesis runtime. • routability – Performs placement-aware congestion reduction techniques to produce a routable netlist. There is a trade-off in timing QoR to improve synthesis runtime.
-use_fsm_explorer 1 0	<p><i>Synplify Pro, Synplify Premier</i></p> <p>Enables/disables the FSM Explorer. See Running the FSM Explorer, on page 589 in the <i>User Guide</i> for information about using it.</p>

Xilinx CPLD Technologies

The following table lists the supported Xilinx CPLD technologies and corresponding parameter values for `set_option -technology`.

Technology	Parameter
CoolRunner-II	coolrunner2
CoolRunner-II Automotive	acoolrunner2
CoolRunner XPLA3	coolrunner

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 453](#)).

This section provides information on specific options for the project command for the Xilinx CPLD technologies. See [project, on page 111](#) 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.

Xilinx Output Files and Forward Annotation

After synthesis, the software generates a log file and output files for Xilinx designs. The following describe some of the reports with Xilinx-specific information, or files that forward-annotate information to the Xilinx place-and-route tool.

- [Xilinx Resource Usage Reports](#), on page 941
- [Xilinx Net Buffering Report](#), on page 942
- [Virtex-5 and Virtex-6 Detailed Clock Path Report](#), on page 942
- [Custom Timing Reports for Xilinx](#), on page 943
- [Forward Annotating Xilinx Output](#), on page 944
- [Forward-Annotation of Constraints for Xilinx DSP Inferencing](#), on page 945

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.

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 579](#) 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 Name	Type	Pin Name	Pin Dir	Delay	Arrival Time	No. of Fan Out(s)
<hr/>						
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 :	<code>adcm clk1_1_derived_clock</code>					

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: <code>adcm clk1_1_derived_clock</code>						
=====						
Starting Points with Worst Slack						

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
in2_reg[0]	<code>adcm clk1_1_derived_clock</code>	FD	Q	<code>in2_reg[0]</code>	0.450	0.242
in2_reg[1]	<code>adcm clk1_1_derived_clock</code>	FD	Q	<code>in2_reg[1]</code>	0.450	0.242
in2_reg[2]	<code>adcm clk1_1_derived_clock</code>	FD	Q	<code>in2_reg[2]</code>	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.

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 482](#) in the *User Guide* for a procedure.

Forward Annotating Xilinx Output

The following procedures show you how to pass information to the place-and-route tool; this information generally has no impact on synthesis. Typically, you use attributes to pass this information to the place-and-route tools. This section describes the following:

- [Specifying Pin Locations](#), on page 944
- [Passing Technology Properties](#), on page 944
- [Specifying Padtype and Port Information](#), on page 945

Specifying Pin Locations

In certain technologies you can specify pin locations that are forward annotated to the corresponding place-and-route tool. The following procedure shows you how to specify the appropriate attributes.

1. Start with a design using an appropriate Xilinx technology.
2. Add the appropriate attribute to the port. For a bus, list all the bus pins, separated by commas.
 - To add the attribute from the SCOPE interface, click the Attributes tab and specify the appropriate attribute and value.
 - To add the attribute in the source files, use the appropriate attribute and syntax. For details about the attributes in the tables, see the *Attribute Reference Manual*.

Vendor Family	Attribute and Value
Xilinx	<code>syn_loc {pin_number}</code> or <code>xc_loc {pin_number}</code> See Specifying RLOCs , on page 912 for details about relative placement.

Passing Technology Properties

The following table summarizes the attribute used to pass technology-specific information for certain Xilinx devices.

Vendor	Attribute for passing properties
Xilinx	Specify the Xilinx properties directly in the source code. The software passes them to the place-and-route tool. For example: attribute INIT of RAM1: label is "0000"; or /* synthesis INIT_xx = "value" */

Specifying Padtype and Port Information

For certain devices, you can use the following attribute to specify technology-specific port information or padtype.

Vendor	Attribute
Xilinx	<code>xc_padtype</code> <code>define_attribute {a[3:0]} xc_padtype {IBUF_GTL}</code>
Xilinx	<code>xc_pullup/xc_pulldown</code> <code>define_attribute {port_name} xc_pulldown {1}</code>

Forward-Annotation of Constraints for Xilinx DSP Inferencing

Xilinx Virtex-5 and Newer Technologies

The synthesis tool automatically creates a new FDC file for DSPs inferred in the design. This DSP constraint file can be used to achieve similar DSP mappings for subsequent synthesis runs of the design in the tool.

To forward-annotate constraints for Xilinx DSP inferencing, do the following:

1. Run synthesis for the design.
 - An FDC file is created in the top-level implementation results directory called
`designName_dsp.fdc`
 - All instances that infer DSPs in the design are written to the FDC file using the `syn_DSPstyle` attribute as shown below:

```
define_attribute {i:a} {syn_DSPstyle} {dsp48}
```

```
define_attribute {i:a} {syn_DSPstyle} {simd}
```

The attribute is never written with the value of logic in this FDC file.

2. Copy the *designName_dsp.fdc* file or rename it, and then add the file to the project.
3. Rerun the design to achieve DSP inferencing stability as occurred in the original design.

Compile Point Flow

In ACP mode, separate FDC files are created in each compile point directory. Once ACP completes, all the FDC files are merged into the top-level FDC file as generated above.

Note: DSP instances must be specified with RTL names in the FDC file.

Limitations

Limitations for this feature include the following:

- Instance names must be the same from release to release.
- Registers packed/unpacked into the DSP can change.
- DSP mapping can change for timing driven dependencies.
- DSP mapping can change depending on the thresholds used for mapping operator instances.

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 947
- [Xilinx Xflow and Xtclsh](#), on page 947
- [Xilinx Incremental Flow](#), on page 948
- [Compile Point Remapping in Xilinx Designs](#), on page 920
- [ISE to Synthesis Flow](#), on page 948
- [Reoptimizing with EDIF Files](#), on page 949
- [Running Post-Synthesis Simulation](#), on page 950

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 1102 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.

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)

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 1139](#) 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 1142](#) 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 1147](#) in the *User Guide* for a procedure.

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 626](#) of the *User Guide* and [Xilinx Partition Flow, on page 1142](#) 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 761](#) in the *User Guide* and [ise2syn, on page 95](#) for details.

Reoptimizing with EDIF Files

You can add an EDIF file created by the Synopsys FPGA synthesis software and resynthesize, to further refine and optimize your design. The EDIF can be resynthesized as a lower-level module, or as the top level of the hierarchy.

Incorporating EDIF for Lower-Level Module

The design can include EDIF netlists for IP, where the EDIF is generated outside Xilinx. For example, the Synplify tools could be used to create EDIF for an IP module from the original RTL netlist. To add a Synplify EDIF file as a lower-level module in a design, follow the procedure below. The procedure can be used for an initial run or for reoptimization.

1. Do the following when generating the EDIF for the submodule:
 - Disable I/O insertion, as the module will not be instantiated as the top level of the hierarchy.
 - Make sure the EDIF file name matches the module name.
2. Create a wrapper for the generated EDIF.
 - Create a Verilog or VHDL stub file, which contains only the module and port declarations. For example:

```
module myBB ( a, b, clk ) /* synthesis syn_black_box=1 */
  input a;
  output b;
  input clk;
```

- Declare it a black box by adding the `syn_black_box` directive after the closing parenthesis and before the semicolon:

```
module myIP ( ..... ) /* synthesis syn_black_box = 1 */;
```

Make sure that there is no other logic in the file for the black box.

3. Create a project and add the following files: the top-level module, the EDIF file for the IP, and the black box stub file.
4. Synthesize or resynthesize the design.

The tool reads in the black box and the EDIF file for the submodule when it synthesizes the design.

Incorporating EDIF for a Top-Level Module

To resynthesize an EDIF file created by the Synplify Pro or Synplify Premier tool at the top level of the hierarchy, follow the steps below:

1. Make sure your design does not include any mixed-language files.
2. Check that the EDIF file name matches the module name.
3. Create a project and add the EDIF file to the design.
4. Specify the EDIF file as the top-level design module by doing the following:
 - Click Implementation Options and go to the Verilog or VHDL tab.
 - Enter the module name in the Top Level Module/Entity field. If your module is not in the work library, first specify the library as *libraryName.moduleName*.
 - Click OK.
5. Resynthesize your design.

Running Post-Synthesis Simulation

For post-synthesis simulation with a Xilinx design, do the following:

1. Run synthesis as usual.

The run generates a `vhm` file, which references the `synplify` library:

```
library synplify;  
use synplify.components.all;  
library UNISIM;  
use UNISIM.VCOMPONENTS.all;
```

2. Set up the libraries.
 - Create a library called `synplify` and compile `synplify.vhd` into it. The `synplify.vhd` file is located in `installDirectory/lib/vhdl_sim`.
 - Create a library called `UINISIM`, and compile the `UNISIM` simulation library provided by Xilinx into it.
3. Compile the `vhm` file into `work`. For example:

```
vcom -work synplify installDirectory/lib/vhdl_sim/synplify.vhd
```

Xilinx Attribute and Directive Summary

The following table summarizes the synthesis and Xilinx-specific attributes and directives available with the Xilinx Technology.

Attribute/Directive	Description
<code>black_box_pad_pin</code>	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.
<code>black_box_tri_pins</code>	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.
<code>CLOCK_DEDICATED_ROUTE</code>	Determines if the software enforces placement rules for the specified architecture.
<code>diff_term</code>	Specifies Differential Termination for true differential input I/O standards.
<code>full_case</code>	Specifies that a Verilog case statement has covered all possible cases.
<code>loop_limit</code>	Specifies a loop iteration limit for for loops.
<code>parallel_case</code>	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
<code>syn_allow_retiming</code>	Specifies if registers can be moved during retiming.
<code>syn_allowed_resources</code>	Specifies allowed resources for compile points. The value assigned to a given compile point includes the resources used by its children (at all levels).
<code>syn_assign_to_region</code>	Assigns logic to regions created on the device.
<code>syn_assign_to_slr</code>	Available for <i>Xilinx Virtex-7 Stacked Silicon Interconnect (SSI)</i> 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.

Attribute/Directive	Description
<code>syn_async_reg</code>	Controls the minimum number of registers used to synchronize a signal across two asynchronous clock domains.
<code>syn_auto_insert_bufg</code>	Use this attribute to automatically insert a BUFG primitive on high fanout nets.
<code>syn_auto_insert_bufgmux</code>	Controls global automatic BUFGMUX insertion
<code>syn_black_box</code>	Defines a black box for synthesis.
<code>syn_clean_reset</code>	Changes asynchronous reset registers, which cannot go into DSP blocks, into synchronous reset logic.
<code>syn_clock_gmux_proxy</code>	Allows the software to automatically generate proxy clocks at the output of the BUFGMUX.
<code>syn_clock_priority</code>	Establishes clock priority between clocks. The priority is forward-annotated in the UCF file.
<code>syn_diff_io</code>	Controls the inference of I/O buffers.
<code>syn_direct_enable</code>	Identifies which signal to use as the enable input to an enable flip-flop when multiple candidates are possible.
<code>syn_direct_reset</code>	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.
<code>syn_direct_set</code>	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.
<code>syn_disable_purifyclock</code>	Disables the purification of clocks.
<code>syn_DSPstyle</code>	Determines if an operator, register, or module/architecture is placed in the DSP component.

Attribute/Directive	Description
<code>syn_edif_bit_format</code>	Controls the character formatting and style of bus signal names, port names, and vector ranges in the EDIF output file.
<code>syn_edif_scalar_format</code>	Controls the character formatting of scalar signal and port names in the EDIF output file.
<code>syn_encoding</code>	Specifies the encoding style for state machines.
<code>syn_enum_encoding</code>	Specifies the encoding style for enumerated types (VHDL only).
<code>syn_fast_auto</code>	Speeds up signal transitions on output ports.
<code>syn_forward_io_constraints</code>	Enables forward annotation of I/O constraints to the Xilinx ISE Design Suite.
<code>syn_fsm_correction</code>	Specifies a Hamming 3 encoding used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers.
<code>syn_global_buffers</code>	Sets the number of global buffers to use in a design.
<code>syn_hier</code>	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
<code>syn_highrel_ioconnector</code>	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.
<code>syn_insert_buffer</code>	Inserts a clock buffer according to the Xilinx specified values.
<code>syn_insert_pad</code>	Inserts I/O buffers to either ports or nets.
<code>syn_isclock</code>	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
<code>syn_keep</code>	Prevents the internal signal from being removed during synthesis and optimization.
<code>syn_loc</code>	Specifies pin locations for I/O pins and cores, and forward-annotates this information to the ISE Design Suite.

Attribute/Directive	Description
<code>syn_looplimit</code>	Specifies a loop iteration limit for while loops in the design.
<code>syn_macro</code>	Prevents instantiated macros from being merged or otherwise optimized away.
<code>syn_map_dffrs</code>	Allows the synthesis software to handle registers with asynchronous set and reset for Virtex-6 and Spartan-6 devices.
<code>syn_maxfan</code>	Overrides the default fanout guide for an individual input port, net, or register output.
<code>syn_multstyle</code>	Determines implementation style for multipliers.
<code>syn_netlist_hierarchy</code>	Determines if the EDIF output netlist is flat or hierarchical.
<code>syn_noarrayports</code>	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.
<code>syn_noclockbuf</code>	Controls the automatic insertion of global clock buffers.
<code>syn_no_compile_point</code>	Ignores modules as compile points in the Automatic Compile Point flow.
<code>syn_noprune</code>	Controls the automatic removal of instances that have outputs that are not driven.
<code>syn_pad_type</code>	Specifies an I/O buffer standard for Virtex-II and later, and Spartan-3 and later technologies
<code>syn_pipeline</code>	Specifies that registers be moved into ROMs or multipliers to improve frequency.
<code>syn_preserve</code>	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
<code>syn_probe</code>	Adds probe points for testing and debugging.
<code>syn_radhardlevel</code>	Enables the triple modular redundancy and voting logic in the design.
<code>syn_ramstyle</code>	Determines the way in which RAMs are implemented.

Attribute/Directive	Description
<code>syn_reduce_controlset_size</code>	Controls the minimum size of the unique control-set on which control-set optimizations can occur.
<code>syn_reference_clock</code>	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
<code>syn_replicate</code>	Disables replication.
<code>syn_resources</code>	Specifies the resources used inside a black box.
<code>syn_romstyle</code>	Determines how ROM architectures are implemented.
<code>syn_rw_conflict_logic</code>	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.
<code>syn_safe_case</code>	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.
<code>syn_safefsm_pipe</code>	Removes the pipeline register on the error recovery path for the Preserve and Decode Unreachable States option.
<code>syn_sharing</code>	Specifies resource sharing of operators.
<code>syn_shift_resetphase</code>	Allows you to remove the flip-flop on the inactive clock edge, built by the reset recovery logic for an FSM when a single event upset (SEU) fault occurs.
<code>syn_slow</code>	Increases transition time of a specific output port or ports.
<code>syn_srl_mindepth</code>	Lets you specify the threshold number of registers to pack into an SRL.
<code>syn_srlstyle</code>	Determines how to implement the sequential shift (seqShift) components. This attribute applies to Virtex technologies.
<code>syn_state_machine</code>	Determines if the FSM Compiler extracts a structure as a state machine.

Attribute/Directive	Description
<code>syn_tco<n></code>	Defines timing clock to output delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tpd<n></code>	Specifies timing propagation for combinational delay through the black box. The <i>n</i> indicates a value between 1 and 10.
<code>syn_tristate</code>	Specifies that a black-box pin is a tristate pin.
<code>syn_tristatetomux</code>	Converts tristate drivers that drive nets below a certain limit to multiplexers.
<code>syn_tsu<n></code>	Specifies the timing setup delay for input pins, relative to the clock. The <i>n</i> indicates a value between 1 and 10.
<code>syn_useenables</code>	Prevents generation of registers with clock enable pins.
<code>syn_useioff</code>	Packs flip-flops in the I/Os to improve input/output path timing.
<code>syn_user_instance</code>	Prevents user-instantiated instances from being merged or otherwise optimized away.
<code>syn_vote_loops</code>	Specifies whether or not to add any synchronous voter to a sequential feedback path for a distributed TMR module.
<code>syn_vote_register</code>	Enables voter logic to be inserted after every register of the module specified for distributed TMR.
<code>translate_off/translate_on</code>	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
<code>xc_area_group</code>	Assigns a compile point to a Xilinx area group.
<code>xc_clockbuftype</code>	Specifies that a clock port use the Clock Delay Locked Loop primitive, CLKDLL.
<code>xc_fast</code>	Speeds up the transition time of the output driver.
<code>xc_fast_auto</code>	Controls the instantiation of fast output buffers in Virtex designs.

Attribute/Directive	Description
<code>xc_global_buffers</code>	Controls the number of global buffers to use in a design.
<code>xc_loc</code>	Specifies the location (placement) of ports.
<code>xc_map</code>	Specifies RLOCs. RLOCs are relative location constraints.
<code>xc_padtype</code>	Specifies an I/O buffer standard in Virtex designs.
<code>xc_props</code>	Used to specify the Xilinx attributes to forward-annotate to the gate-level netlist.
<code>xc_pullup/xc_pulldown</code>	Specifies that a port is a pullup or pull-down port.
<code>xc_rloc</code>	Specifies the relative locations of all instances with the same <code>xc_use</code> attribute value.
<code>xc_slow</code>	<p><i>CPLD Devices</i></p> <p>Specifies a slow transition time for the driver of an output port.</p> <p>The Synplify Premier software does not use this attribute.</p>
<code>xc_use_keep_hierarchy</code>	Preserves the hierarchy of the marked module for diagnostics and timing simulation.
<code>xc_use_timespec_for_io</code>	Uses the TIMESPEC command to write out I/O constraints.
<code>xc_use</code>	Assigns a group name to component instances.

Index

Symbols

.est.srr file 192
.ta.srm file 197
.adc file 182
.areasrr file 191
.edf file 886
.est file 192
.fse file 192
.info file 192
.ini file 183
.lpf file 668, 670
.ndf file 886
.nrf file 183
.opt file
 input 183
 output 193
.prj file 183
.sap
 annotated properties for analyst 194
.sar file 195
.sdc file 183
.sfp file 184
.srd file 195
.srm file 195
.srp file 195
.srr file 199
 watching selected information 54
.srs file 195
 initial values (Verilog) 316
.sv file 184
 SystemVerilog source file 185
.ta file
 See timing report file 197
.tcl file
 input 184

 output 197
tcl file
 See also Tcl files 197
.v file 185
.vhd file 185
.vhm file 198, 540, 950
.vm file 198, 541
.xnf file (Xilinx). *See* xnf file (Xilinx)

A

Achronix
 attributes 542
 directives 542
 initial values 495
 Speedster22iHD 531
 technologies 471
ACTgen macros 826
adc file (analysis design constraint) 182
adder
 SYNCore 418
adders
 SYNCore 419
 wide. *See* wide adders/subtractors.
Allow Docking command 55
alspin
 bus port pin numbers 822
Alt key, selecting columns in Text Editor 64
altpll
 component declaration files 640
 constraints 641
 using 640
analysis design constraint file (.adc) 182
Analyst toolbar 78
annotated properties for analyst
 .sap 194
 .timing annotated properties (.tap) 197

- archive file (.sar) 195
area estimation file (.est) 192
areasrr file
 hierarchical area report 215
Arria 621
 device options 621
 file format options 624
 set_option device options 623
 set_option synthesis options 624
arrow keys, selecting objects in Hierarchy Browser 129
arrow pointers for push and pop 128
asymmetric RAM
 inferring 269
 instantiating as a black box 296
asymmetric RAM examples 269
asynchronous clock report
 description 213
ATOM mapping, Intel 555
attributes
 inferring RAM 231
 Intel 660
attributes (Achronix) 542
attributes (Microsemi) 828
attributes (Xilinx) 951
Attributes demo 68
auto constraints 179
 Maximize option 94
- B**
- black box instantiation
 true dual-port asymmetric RAM 296
black boxes
 See also macros, macro libraries
 for IP cores 886
 for predefined macros (Intel) 554
 Lattice 683, 703
 Microsemi 763
 specifying timing information for Xilinx cores 886
 Xilinx 869
 Xilinx macros 866
- block RAM
 dual-port RAM examples 246
 inferring 233
- mapping ROM (Xilinx) 857
modes 229
NO_CHANGE mode example 242
RAM with control signals examples 258
READ_FIRST mode example 241
single-port RAM examples 243
types 229
WRITE_FIRST mode example 239
- block RAMs
 inferring byte enables examples 302
 Lattice 672
 parity bus, Xilinx 856
 syn_ramstyle attribute 830
 Xilinx single port and dual-port 856
- blocks
 packing I/O (Xilinx) 867
- buffering, Intel 618
- BUFGDLL 908
- BUFGMUX_1 inference 906
- BUFGMUX/BUFGMUX_1 inference 868, 906
- BUFR clock buffers 907
- bus constraints
 multicycle (Intel) 654
- buses
 INIT values for bits 896
 RLOC values for bits 914
- buttons and options, Project view 92
- byte-enable RAM
 inferring 302
- byte-enable RAM examples 302
- byte-enable RAMs
 SYNCORE 388
- byte-wide write enable RAM
 Intel 560
- C**
- carry chains
 inferring 709
- cck.rpt file (constraint checking report)
 191
- check boxes, Project view 92
- Clearbox
 features 638
 primitives 639

- cliquing 626
 clock buffering report, log file (.srr) 202
 clock buffers 908
 clock DLLs 908
 clock enables
 inferring registers (Intel) 556
 inferring registers (Xilinx) 858
 clock groups
 Clock Relationships (timing report) 211
 clock pin drivers, selecting all 102
 clock relationships, timing report 211
 clock report
 asynchronous 205
 clock skew example (Synplify Premier)
 910, 911
 clock skew, Xilinx 921
 Clock Tree, HDL Analyst tool 102
 clocks
 asynchronous report 213
 declared clock 207
 defining 102
 derived clock 208
 inferred clock 207
 system clock 208
 color coding
 Text Editor 64
 commenting out code (Text Editor) 64
 companion parts, Stratix, HardCopy 616
 compile point constraints
 forward-annotating (Cyclone, Arria) 655
 compile points
 Intel 619
 Lattice 712
 Microsemi 824
 update compile point timing data 528
 updating data (Microsemi) 809
 Xilinx 920
 Xilinx flow 948
 compiler report, log file (.srr) 200
 Constraint Check command 218
 constraint checking report 218
 constraint files 145
 .sdc 183
 automatic. *See* auto constraints
 fdc and sdc precedence order 148
 forward-annotation, Lattice
 474, 668, 762, 836
 Intel Max+Plus II 654
 Microsemi 820
 PLL 556
 constraint files (.sdc)
 creating 77
 constraint priority 148
 constraints
 altplls 641
 auto constraints. *See* auto constraints
 non-DC 156
 priority 148
 report file 218
 styles 147
 types 144
 context help editor 65
 context of filtered schematic, displaying
 134
 context sensitive help
 using the F1 key 30
 control panel
 displaying instances 466
 displaying nets 466
 displaying row sites 466
 displaying signal pins 466
 Physical Analyst view 465, 467
 control panel (Physical Analyst) 465, 467
 control-sets, Xilinx 878
 copying
 for pasting 85
 core voltage switch 616
 core voltage, Stratix III 616, 645
 CoreGen 886
 cores, instantiating in Xilinx designs 886
 counter compiler
 SYNCORE 442
 counters
 SYNCORE 443
 CPLD technology
 Xilinx 935
 critical paths 139
 analyzing 140
 finding 140
 cross-clock paths, timing analysis 211

cross-hair mouse pointer 74
crossprobing 119
 definition 119
Ctrl key
 avoiding docking 76
 multiple selection 73
 zooming using the mouse wheel 75
customer support, Synopsys 32
cutting (for pasting) 78
Cyclone 621
 device options 621
 file format options 624
 set_option device options 623
 set_option synthesis options 624

D

DDR register inference
 Xilinx 860
declared clock 207
define_clock
 forward-annotation, Intel Max+Plus II 654
 forward-annotation, Microsemi 820
define_false_path
 forward-annotation, Microsemi 821
define_input_delay
 forward-annotation, Intel Max+Plus II 654
define_multicycle_path
 forward-annotation, Microsemi 821
define_output_delay
 forward-annotation, Intel Max+Plus II 654
define_path_delay
 forward-annotation, Microsemi 821
deleting
 See removing
Demos & Examples 67
derived clock 208
design flows
 synthesis (Intel) 550
 synthesis (Xilinx) 838
Design Hierarchy view (hierarchical project management) 39
Design Hierarchy view (hierarchical project management) icons 40
Design Plan Editor 461
Design Plan Editor view 461
design plan file (.sfp) 184
Design Plan Hierarchical view 459
Design Plan Hierarchy view 459
Design Plan view 461
design plan view file (.srp) 195
design planner constraint files (.sfp)
 creating 77
Design Planner option views 458
Design Planner view 458
design size, schematic sheet setting 122
device option
 Update Compile Point Timing Data 528
device options 529, 730
 CPLDs (Xilinx) 935
 iCE65 Lattice technology 730
 Spartan technologies (Xilinx) 925
 Virtex technologies (Xilinx) 925
device options (Microsemi) 813
device view
 Physical Analyst 469
directives
 Intel 660
directives (Achronix) 542
directives (Microsemi) 828
directives (Xilinx) 951
disable I/O insertion (Lattice) 711
Disable Sequential Optimizations (Xilinx) 919
Dissolve Instances command 137
distributed RAM
 inferring 264
distributed RAM examples 267
docking 55
 avoiding 76
docking GUI entities
 toolbar 76
DSP blocks

- inferencing 765
 - Intel megafunctions 567
 - packing resets (Intel) 592
 - DSP blocks (Intel) 567
 - DSP48 (Xilinx) 839
 - DSP48 structures (Xilinx) 839
 - DSP48 Xilinx
 - inferring XNOR 847
 - symmetric rounding 847
 - DSP48E, wide adder inference 877
 - dual-port RAM
 - true dual-port RAM
 - See* multi-port RAMs
 - dual-port RAM examples 246
 - dual-port RAMs
 - SYNCORE parameters 371
- E**
- EDIF
 - structural, for Xilinx IP cores 886
 - EDIF files
 - reoptimizing 949
 - edif files
 - Intel naming requirements 658
 - editor view
 - context help 65
 - enable prepacking 922
 - enable_io_map.txt file 557
 - encoding
 - state machine
 - FSM Explorer 94
 - est file 192
 - estimation file, area (.est) 192
 - estimation log file (_est.srr) 192
 - examples
 - Demos & Examples 67
 - Interactive Attribute Examples 68
 - Explorer, FSM
 - enabling 94
- F**
- failures, timing (definition) 141
 - fanin (Lattice) 715
 - fanout
 - Microsemi 804
 - Fanout Guide option (Xilinx) 917
 - fanouts (Intel) 617
 - fanouts (Lattice) 711
 - fanouts (Xilinx) 916
 - fdc
 - constraint priority 148
 - precedence over sdc 148
 - fdc constraints 150
 - generation process 148
 - fdc file
 - relationship with other constraint files 145
 - feature comparison
 - FPGA tools 23
 - FIFO compiler
 - SYNCORE 334
 - FIFO flags
 - empty/almost empty 356
 - full/almost full 355
 - handshaking 356
 - programmable 358
 - programmable empty 361
 - programmable full 359
 - FIFOs
 - compiling with SYNCORE 335
 - file format options
 - project command (Intel) 636
 - files
 - _est.srr 192
 - .adc 182
 - .areasrr 191
 - .cdc 182
 - .est 192
 - .fdc 182
 - .fse 192
 - .info 192
 - .ini 183
 - .lpf 668, 670
 - .nrf 183
 - .opt 183, 193
 - .prj 183
 - .sar 195
 - .sdc 183
 - .sfp 184
 - .srm 195, 197

- .srp 195
- .srr 199
 - watching selected information 54
- .srs 195
- .ta 197
- .tcl for Xilinx xtclsh 184, 197
- .v 184, 185
- .vhd 185
- .vhm 198
- .vm 198
- .xnf (Xilinx). *See* xnf file (Xilinx)
- altpll component declarations 640
- area estimation (.est) 192
- compiler output (.srs) 195
- constraint (.adc) 182
- constraint (.sdc) 183
- creating 77
- customized timing report (.ta) 197
- design component info (.info) 192
- design plan (.sfp) 184
- design plan view (.srp) 195
- estimation log (.est.srr) 192
- initialization (.ini) 183
- log (.srr) 199
 - watching selected information 54
- mapper output (.srm) 195, 197
- Netlist Restructure (.nrf) 183
- output
 - See* output files
- partitioned RTL view (.srp) 195
- project (.prj) 183
- RTL view (.srs) 195
- srr 199
 - watching selected information 54
- state machine encoding (.fse) 192
- Synopsys archive file (.sar) 195
- synthesis output 191
- Technology view (.srm) 195, 197
- Verilog (.v) 184, 185
- VHDL (.vhd) 185
- Xilinx (.opt) 183, 193
- files for synthesis 182
- filtered schematic
 - compared with unfiltered 105
- filtering 133
 - commands 133
 - compared with flattening 137
 - FSM states and transitions 105
 - paths from pins or ports 141
- filtering critical paths 140
- finding
 - critical paths 140
 - information on synthesis tool 31
 - GUI 30
- finite state machines
 - See* state machines
- Flatten Current Schematic command 137
- Flatten Schematic command 137
- flattening
 - commands 135
 - compared with filtering 137
 - selected instances 136
- Float command
 - Watch window popup menu 55
- floating
 - toolbar 76
- floating toolbar popup menu 76
- forward annotation
 - frequency constraints 541
 - frequency constraints in Xilinx 880
 - initial values 316
- Forward Annotation of Initial Values
 - Verilog 316
- forward-annotation
 - compile point constraints (Cyclone, Arria) 655
 - constraints 669
 - Intel RAM init values 316
 - ispLEVER 668
 - Lattice constraints 668
 - Microsemi 820
- frequency
 - cross-clock paths 211
- Frequency (Mhz) option, Project view 93
- fse file 192
- FSM Compiler option, Project view 94
- FSM Compiler, enabling and disabling
 - globally
 - with GUI 94, 737
- FSM encoding file (.fse) 192
- FSM Explorer
 - enabling 94
- FSM Explorer option, Project view 94
- FSM toolbar 80, 81
- FSM Viewer 103

FSMs (finite state machines)
See state machines

G

gated clocks (Lattice iCE60/iCE40) 738
 gated clocks (Xilinx) 919
 generated clocks (Intel) 919
 generated clocks (Lattice) 739
 generic technology library 188
 global buffer promotion
 iCE40 708
 global comments
 initializing Xilinx RAM 891
 global sets/resets
 Xilinx designs 883
 graphical user interface (GUI), overview
 33
 grey boxes
 using 640
 GSR resource (Lattice) 711
 GSR resources 752
 GSR, Xilinx 882
 GTECH library. *See* generic technology library
 gtech.v library 188
 gui
 synthesis software 26
 GUI (graphical user interface), overview
 33

H

HardCopy
 companion parts 616
 HDL Analyst tool 97
 accessing commands 106
 analyzing critical paths 139
 Clock Tree 102
 crossprobing 119
 filtering designs 133
 finding objects 117
 hierarchical instances. *See* hierarchical instances
 object information 108
 preferences 122

push/pop mode 125
 ROM table viewer 327
 schematic sheet size 122
 schematics, filtering 133
 schematics, multiple-sheet 122
 status bar information 108
 Synplify license 97
 title bar information 122
 HDL Analyst toolbar
 See Analyst toolbar
 HDL Analyst views 98
 See also RTL view, Technology view
 HDL files, creating 77
 header, timing report 206
 help
 online
 accessing 30
 hidden hierarchical instances 113
 are not flattened 137
 Hide command
 floating toolbar popup menu 76
 Log Watch window popup menu 55
 Tcl Window popup menu 58
 hierarchical area report 215
 .areasrr file 215
 hierarchical instances 111
 compared with primitive 110
 display in HDL Analyst 111
 hidden 113
 opaque 111
 transparent 111
 hierarchical project management views
 38
 hierarchical schematic sheet, definition
 122
 hierarchy
 flattening
 compared with filtering 137
 pushing and popping 125
 schematic sheets 122
 Hierarchy Browser 129
 changing size in view 98
 Clock Tree 102
 finding schematic objects 117
 moving between objects 102
 RTL view 98
 symbols (legend) 103

-
- Technology view 100
 - trees of objects 102
-
- I
- I/O insertion 527, 751
 - Arria 622
 - Intel Cyclone 622
 - Intel Stratix 622
 - VHDL manual (Xilinx) 905
 - I/O insertion (Intel) 616
 - I/O insertion (Microsemi) 808
 - I/O insertion (Xilinx) 867
 - I/O locations
 - assigning automatically (Xilinx) 900
 - manually assigning (Xilinx) 905
 - I/O pads
 - instantiating (Xilinx) 867
 - I/O standards
 - QSF constraints 161
 - I/Os
 - packing in Intel designs 643
 - packing in Xilinx designs 897
 - preserving 752
 - specifying pad type (Xilinx) 911
 - IBUFDS
 - inference 907
 - IBUFGDS
 - inference 907
 - iCE65 Lattice device options 730
 - Identify Instrumentor
 - launching 83
 - IEEE 1364 Verilog 95 standard 186
 - Implementation Directory 49
 - Implementation Results 49
 - indenting a block of text 64
 - indenting text (Text Editor) 64
 - inference
 - BUFGMUX/BUFGMUX_1 906
 - Xilinx BUFGMUX/BUFGMUX_1 868, 906
 - Xilinx I/O buffers 867, 906
 - inferencing
 - DSP blocks 765
 - inferred clock 207
 - info file (design component info) 192
 - ini file 183
 - INIT property
 - initializing Xilinx RAMs, Verilog 890
 - initializing Xilinx RAMs, VHDL 893
 - specifying with attributes 894
 - INIT values
 - Xilinx registers 895
 - initial value data file
 - Verilog 313
 - Initial Values
 - forward annotation 316
 - initial values
 - \$readmemb 310
 - \$readmemh 310
 - Achronix 495
 - Lattice ICE65/iCE40 684
 - initial values (Verilog)
 - netlist file (.srs) 316
 - initialization file (.ini) 183
 - input files 182
 - .adc 182
 - .ini 183
 - .nrf 183
 - .opt 183, 193
 - .sdc 183
 - .sfp 184
 - .sv 185
 - .v 184, 185
 - .vhd 185
 - inserting
 - bookmarks (Text Editor) 64
 - instances
 - hierarchical
 - dissolving 130
 - making transparent 130
 - hierarchical. *See* hierarchical instances
 - primitive. *See* primitive instances
 - instances (Physical Analyst) 466
 - Intel
 - asymmetric RAM 560
 - attributes 660
 - cliquing. *See* cliquing.
 - constraints 654
 - device support 548
 - device support. *See also* Intel technologies 548

- directives 660
 DSP block inference 567
 edif and vqm filenames 658
 fanout limits 617
 forward annotation 550
 forward-annotation, Max+Plus II 654
 grey boxes *See* grey boxes
 I/O packing 643
 inferring LUTRAMs 252
 IP core support 552
 LUTRAMs 254
 macro library, Verilog 552
 macro library, VHDL 552
 macros 554
 mapping to ATOMs 555
 Max+Plus II forward-annotated constraints 654
 packing I/Os 643
 pad cell insertion 557
 PLLs 556
 PLLs. *See* altplls
 prepared components method 594
 RAM inference 558
 RAM with control signals 258
 RAMs
 inference (Stratix) 558
 register inference 556
 reports 649
 ROM inference 565
 simulating LPMS 646
 SOPC Builder 158
 specifying VHDL library 552
 technology keywords 636
- Intel black boxes
 pad cell insertion (Intel) 557
- Intel Clearbox
 megafunctions 638
- Intel device options
 Arria 621
 Cyclone 621
 MAX 628
 MAX II 625
 Stratix 621
- Intel file format options
 MAX 631
- Intel Mapper
 new 637
- Intel megafunctions
 DSP blocks 567
- Intel New Mapper
 using 637
- Intel RAM
 byte-wide write enable 560
- Intel RAMs
 \$readmemb/\$readmemh initialization 316
 inference (Arria) 558
 inference (Cyclone) 558
- Intel technologies
 MAX 628
- Interactive Attribute Examples 68
- interface information, timing report 212
- IOBUFDS
 inference 907
- IP cores 886
- IP encryption
 Lattice 744
- IP encryption (Lattice) 744
- IPs
 SYNCORE byte-enable RAMs 388
 SYNCORE counters 443
 SYNCORE FIFOs 335
 SYNCORE RAMs 365
 SYNCORE ROMs 405
 SYNCORE subtractors 419
- isolating paths from pins or ports 141
- ispLEVER
 forward-annotating constraints for 669
- ## K
- keyboard shortcuts 84
 arrow keys (Hierarchy Browser) 129
- keyword completion, Text Editor 64
- keywords
 completing in Text Editor 64
- ## L
- latches
 in timing analysis 139
 mapping (Xilinx) 866
- Lattice
 attributes and directives 754
 black boxes 683, 703

- Block RAM Support [672](#)
- disable I/O insertion [711](#)
- fanin limit [715](#)
- forward annotation [668, 762, 836](#)
- forward-annotation constraints [668](#)
- generated clocks [739](#)
- GSR resource [711](#)
- IP encryption [744](#)
- macro libraries [703](#)
- macros [683, 703](#)
- map logic to macrocells [714](#)
- maximum cell fanin [715](#)
- maximum terms/macrocell [715](#)
- percentage of design to optimize for timing [715](#)
- PICs [744](#)
- pipelining [737](#)
- product families [666](#)
- reports [740](#)
- retiming [738](#)
- updating compile points [712](#)
- Lattice iCE65/iCE40
 - gated clocks [738](#)
 - initial values [684](#)
- Lattice iCE65/iCE40 options [731](#)
- Lattice netlist [667](#)
- Lattice options
 - LatticeECP2 and LatticeECP/EC [717](#)
 - MachXO [721](#)
 - ORCA [715](#)
- Launch Identify Instrumentor icon [83](#)
- LCELLS [629](#)
- legacy sdc file. *See* sdc files, difference between legacy and Synopsys standard
- lib2syn
 - using [189](#)
- libraries
 - general technology [187](#)
 - macro, built-in [185](#)
 - post-synthesis simulation [541](#)
 - technology-independent [187](#)
 - VHDL
 - attributes and constraints [186](#)
 - Xilinx post-synthesis simulation [950](#)
- linkerlog file [193](#)
- location constraints
 - RLOC_ORIGIN [914](#)
- RLOCs with synthesis attribute [877, 914](#)
- RLOCs with xc_attributes [912](#)
- log file (.srr) [199](#)
 - watching selected information [54](#)
- log file report [199](#)
 - clock buffering [202](#)
 - compiler [200](#)
 - mapper [201](#)
 - net buffering [202](#)
 - resource usage [203](#)
 - retiming [204](#)
 - summary of compile points [203](#)
 - timing [203](#)
- Log Watch Configuration dialog box [56](#)
- Log Watch window [54](#)
 - Output Windows [62](#)
 - positioning commands [55](#)
- LPMs
 - black box method simulation flow [647](#)
 - prepared components (Intel) [594](#)
 - using in Intel simulation flows [646](#)
 - Verilog library simulation flow [648](#)
 - VHDL prepared component simulation flow [648](#)
- LUT6_2 primitives [922](#)
- LUTRAM
 - examples [254](#)
- LUTRAMs [254](#)
- LUTRAMs, inferring [252, 254](#)

M

- macro libraries
 - Intel Verilog [552](#)
 - Intel VHDL [552](#)
 - Lattice [703](#)
- macro libraries (Xilinx) [880](#)
- macros
 - libraries [185](#)
 - MATH18X18 block [764](#)
 - Microsemi [763](#)
 - SIMBUF [764](#)
- macros (Lattice) [683, 703](#)
- macros (Xilinx) [880](#)
- map logic to macrocells (Lattice) [714](#)
- mapper output file (.srm) [195, 197](#)

-
- mapper report
 log file (.srr) 201
 margin, slack 140
 MAX 628
 device options 628
 file format options 631
 set_option device options 630
 set_option synthesis options 630
 MAX II
 file format options 628
 set_option device options 627
 set_option synthesis options 627
 maximum terms/macrocell (Lattice) 715
 megafunctions
 altplls 640
 Clearbox 638
 grey boxes 640
 Megawizard
 altplls 640
 message viewer
 description 58
 Messages Tab 58
 Microsemi
 ACTgen macros 826
 attributes 828
 black boxes 763
 compile point synthesis 824
 compile point timing data 809
 device options 813
 directives 828
 features 762
 forward-annotation, constraints 820
 I/O insertion 808
 macro libraries 825
 macros 763
 MATH18X18 block 764
 Operating Condition Device Option 811
 output netlist 761
 pin numbers for bus ports 822
 product families 760
 reports 823
 retiming 809
 SIMBUF macro 764
 Tcl implementation options 814
 Microsemi implementing RAM 772
 MIF files
 missing 567
 mouse button operations 73
 mouse operations 71
 Mouse Stroke Tutor 72
 mouse wheel operations 75
 Move command
 floating toolbar window 76
 Log Watch window popup menu 55
 Tcl window popup menu 58
 moving between objects in the Hierarchy Browser 102
 moving GUI entities
 toolbar 76
 multicycle constraints
 forward-annotating 669
 forward-annotation (Intel) 654
 multiple-sheet schematics 122
 multipliers
 DSP blocks 765
 multi-port RAMs
 Xilinx implementations 856
 multisheet schematics
 transparent hierarchical instances 124
- N**
- navigating
 among hierarchical levels
 by pushing and popping 125
 with the Hierarchy Browser 129
 among the sheets of a schematic 122
 nesting design details (display) 130
 net buffering report, log file 202
 netlist file 198
 initial values (Verilog) 316
 Netlist Restructure file (.nrf) 183
 netlists for different vendors 473, 667, 761, 835
 nram
 Xilinx implementations 856
 nrf file 183
- O**
- object information
 status bar, HDL Analyst tool 108

- viewing in HDL Analyst tool 108
- objects
crossprobing 119
dissolving 130
making transparent 130
- objects, schematic
See schematic objects
- OBUFDS
inference 907
- OBUFTDS
inference 907
- OFFSET constraints
unsupported 174
- Online help
F1 key 30
- online help
accessing 30
- opaque hierarchical instances 111
are not flattened 137
- opt file 183, 193
- options
Achronix Speedster22iHD 531
CPLDs (Xilinx) 937
Intel Cyclone 621
Intel Stratix 624
Lattice iCE65/iCE40 731
LatticeSC/SCM and LatticeXP 719
Project view 92
Frequency (Mhz) 93
FSM Compiler 94
FSM Explorer 94
Pipelining 95
Resource Sharing 95
Retiming 95
Spartan (Xilinx) 928
Virtex (Xilinx) 928
- options (Microsemi) 814
- output files 191
.est.srr 192
.areasrr 191
.est 192
.info 192
.sar 195
.srm 195, 197
.srp 195
.srr 199
watching selected information 54
- .srs 195
.ta 197
.vhm 198
.vm 198
.xnf (Xilinx). *See* xnf files (Xilinx). 834
netlist 198
See also files
- Output Windows 62
- overriding FSM Compiler 622, 625
- Overview of the Synopsys FPGA Synthesis Tools 20
- P**
- pad cell insertion file, Intel 557
- pads
disabling I/O insertion 527
disabling I/O insertion (Xilinx) 867
- parameters
SYNCORE adder/subtractor 427
SYNCORE byte-enable RAM 395
SYNCORE counter 449
SYNCORE FIFO 340
SYNCORE RAM 373
SYNCORE ROM 410
parity bus, Xilinx block RAM 856
- partitioned RTL view file (.srp) 195
- partitioning of schematics into sheets 122
- pasting 78
- percentage of design to optimize for timing
Lattice 715
- performance summary, timing report 206
- Physical Analyst
control panel 465, 467
device view 469
input files 467
routing nets 466
- physical constraints
translating QSF constraints 160
- physical optimization
description 22
- PICs 744
- pin location constraints
QSF 160

-
- pin locations
 - specifying (Xilinx) [868](#), [900](#)
 - pins
 - displaying
 - on transparent instances [115](#)
 - displaying on technology-specific primitives [116](#)
 - isolating paths from [141](#)
 - pipelining
 - option [95](#)
 - pipelining (Lattice) [737](#)
 - pipelining (Xilinx) [918](#)
 - Pipelining option, Project view [95](#)
 - Place and Route constraint file (Microsemi) [820](#)
 - PLL constraints [556](#)
 - pointers, mouse
 - cross-hairs [74](#)
 - push/pop arrows [128](#)
 - popping up design hierarchy [125](#)
 - popup menus
 - floating toolbar [76](#)
 - Log Watch window [55](#), [56](#)
 - Log Watch window positioning [55](#)
 - Tcl window [58](#)
 - post-synthesis simulation [540](#)
 - post-synthesis simulation, Xilinx [950](#)
 - precedence of constraint files [148](#)
 - preferences
 - HDL Analyst tool [122](#)
 - prepared components
 - Intel VHDL [594](#)
 - primitive instances [110](#)
 - primitives
 - pin names in Technology view [116](#)
 - prj file [183](#)
 - Process View [50](#)
 - project command
 - CPLDs (Xilinx) [939](#)
 - Intel file formats [636](#)
 - Spartan (Xilinx) [935](#)
 - Virtex (Xilinx) [935](#)
 - XC4000 (Xilinx) [935](#)
 - project files (.prj) [183](#)
 - Project Files view [38](#)
 - project results
 - Implementation Directory [49](#)
 - Process View [50](#)
 - Project Status View [41](#)
 - Project Results View [41](#)
 - Project Status View [41](#)
 - Project toolbar [77](#)
 - Project view [34](#)
 - buttons and options [92](#)
 - options [92](#)
 - Synplify Premier/DP [34](#)
 - Synplify Pro [34](#)
 - Synplify tool [34](#)
 - Project window [34](#)
 - project_name_cck.rpt file [218](#)
 - Promote Global Buffer Threshold (Microsemi) [807](#)
 - push/pop mode, HDL Analyst tool [125](#)
- Q**
- QSF constraints
 - example [161](#)
 - Quartus
 - converting constraints to sdc [160](#)
 - edif and vqm file names [658](#)
 - library versions [552](#)
 - qsf2sdc utility. *See* qsf2sdc [160](#)
 - Quartus megafunctions
 - pad cell insertion [557](#)
- R**
- RAM
 - byte-wide write enable RAM (Intel) [560](#)
 - RAM implementations
 - Microsemi [772](#)
 - RAM inference [229](#)
 - Arria [558](#)
 - Cyclone [558](#)
 - Intel Stratix [558](#)
 - using attributes [231](#)
 - RAM inference (Intel) [558](#)
 - RAM mapping
 - Intel Stratix [559](#)

RAM with control signals examples 258
RAMs
 compiling with SYNCORE 365
 inferring block RAM 233
 initial values (Verilog) 310
 initializing values (Xilinx) 889
 mapping LUTRAMs 252
 multi-port, Xilinx 856
 SYNCORE 365
 SYNCORE, byte-enable 388
RAMs, inferring
 advantages 228
reference manual, role in document set 19
region clock buffers (BUFR) 907
register inference (Xilinx) 858
register packing
 See also syn_useioff attribute 897
 Intel 643
 Xilinx 897
registers
 DDR (Xilinx) 860
 inferring clock enables (Xilinx) 858
 INIT value 895
relative placement. *See* RLOCs
relative position, Xilinx components 877
removing
 bookmark (Text Editor) 64
 window (view) 76
replication
 Intel 618
reports
 constraint checking (cck.rpt) 218
 hierarchical area report 215
 resource usage (Xilinx) 941
 resource usage, Intel 649
 resource usage, Lattice 740
reports (Lattice) 740
resets
 packing in Intel DSP blocks 592
resource sharing 752
Resource Sharing option, Project view 95
resource usage report, log file 203
resourceUsage61 942
retiming
 report, log file 204
 retiming (Lattice) 738
 retiming (Microsemi) 809
 retiming (Xilinx) 918
 Retiming option, Project view 95
RLOC_ORIGINS
 specifying 914
RLOCs 877, 912, 914
 specifying with synthesis attribute 877, 914
 specifying with xc attributes 912
ROM
 block RAM mapping (Xilinx) 857
ROM compiler
 SYNCORE 403
ROM inference (Intel) 565
ROM inference examples 327
ROM initialization
 with rom.info file 330
 with Verilog generate block 331
rom.info file 327
ROMs
 Intel implementations 566
 SYNCORE 405
routing nets (Physical Analyst) 466
RTL view 98
 displaying 79
 file (.srs) 195

S

scan chains
 Intel DSP blocks 570
schematic objects
 crossprobing 119
 definition 108
 dissolving 130
 finding 117
 making transparent 130
 status bar information 108
schematic sheets 122
 hierarchical (definition) 122
 navigating among 122
 setting size 122
schemas

configuring amount of logic on a sheet [122](#)
 crossprobing [119](#)
 filtered [105](#)
 filtering commands [133](#)
 flattening compared with filtering [137](#)
 flattening selectively [136](#)
 hierarchical (definition) [122](#)
 multiple-sheet [122](#)
 multiple-sheet. *See also* schematic sheets
 object information [108](#)
 partitioning into sheets [122](#)
 sheet connectors [109](#)
 sheets
 navigating among [122](#)
 size, setting [122](#)
 size in view, changing [98](#)
 unfiltered [105](#)
 unfiltering [134](#)
SCOPE
 assigning Xilinx pin locations [901](#)
 for legacy sdc [152](#)
 specifying RLOCs [877, 912, 914](#)
sdc
 fdc precedence [148](#)
 SCOPE for legacy files [152](#)
SDC constraints
 from Intel QSF [160](#)
sdc file
 difference between legacy and Synopsys standard [147](#)
 translated UCF constraint examples [164](#)
sdc2fdc utility [154](#)
Search SolvNet
 using [70](#)
select RAM
 initializing [895](#)
selecting
 text column (Text Editor) [64](#)
selecting multiple objects using the Ctrl key [73](#)
sequential optimizations
 disabling [622, 625](#)
 disabling (Intel) [617](#)
 disabling (Xilinx) [919](#)
set_option command
Intel description [631](#)
Intel MAX device syntax [630](#)
Intel MAX II and APEX device syntax
 [627](#)
Intel MAX II and APEX synthesis syntax
 [627](#)
Intel MAX synthesis syntax [630](#)
Intel Stratix, Cyclone, and Arria device syntax [623](#)
Intel Stratix, Cyclone, and Arria synthesis syntax [624](#)
set_rtl_ff_names [156](#)
sheet connectors [109](#)
Shift key [76](#)
shift registers
 Intel definition [556](#)
shortcuts
 keyboard
 See keyboard shortcuts
signal pins (Physical Analyst)
 displaying [466](#)
SIMBUF macro [764](#)
single-port RAM examples [243](#)
single-port RAMs
 SYNCORE parameters [370](#)
sites (Physical Analyst) [466](#)
slack
 cross-clock paths [212](#)
 defined [207](#)
 margin
 definition [141](#)
 setting [140](#)
SLR (Super Logic Regions) [837](#)
SolvNet
 search [70](#)
SOPC2Syn [158](#)
source code
 specifying RLOCs [877, 912, 914](#)
source files
 See also files
 creating [77](#)
srd file [195](#)
SRL tables (Xilinx) [864](#)
srm file [195, 197](#)
srp file [195](#)

- srr file [199](#)
 - watching selected information [54](#)
- srs file [195](#)
 - initial values (Verilog) [316](#)
- Stacked Silicon Interconnect
 - Xilinx technology [837](#)
- standards, supported
 - Verilog [186](#)
 - VHDL [185](#)
- startup block (Xilinx) [882](#)
- state machines
 - encoding
 - displaying [105](#)
 - FSM Explorer [94](#)
 - encoding file (.fse) [192](#)
 - filtering states and transitions [105](#)
 - state encoding, displaying [105](#)
- status bar information, HDL Analyst tool [108](#)
- Stratix [621](#)
 - companion parts [616](#)
 - core voltage for Stratix III [616](#)
 - device options [621](#)
 - file format options [624](#)
 - set_option device options [623](#)
 - set_option synthesis options [624](#)
- structural netlist file (.vhm) [198](#)
- structural netlist file (.vm) [198](#)
- subtracter
 - SYNCore [418](#)
- subtractors
 - SYNCore [419](#)
- summary of compile points report log file (.srr) [203](#)
- Super Logic Regions [837](#)
- supported standards
 - Verilog [186](#)
 - VHDL [185](#)
- symbols
 - Hierarchy Browser (legend) [103](#)
- symmetric rounding
 - DSP48 blocks [847](#)
- syn_dspstyle attribute
 - inferring wide adders/subtractors [884](#)
- syn_edif_bit_format attribute [886](#)
- syn_edif_scalar_format attribute [886](#)
- syn_force_pad attribute
 - using [751](#)
- syn_forward_io_constraints
 - effect on forward-annotation (Intel) [654](#)
- syn_insert_buffer attribute
 - BUFGMUX [906](#)
- syn_keep
 - DSP block inference, Intel [591](#)
 - inferring Lattice PICs [745](#)
- syn_loc attribute
 - translating Intel QSF constraints [161](#)
- syn_maxfan
 - fanout limits (Microsemi) [804](#)
- syn_maxfan attribute
 - Intel [618](#)
 - Xilinx designs [917](#)
- syn_multstyle
 - DSP block inference, Intel [591](#)
- syn_noarrayports attribute
 - use with alspin [822](#)
- syn_preserve
 - DSP block inference, Intel [591](#)
 - preserving registers with INIT values [895](#)
- syn_replicate attribute
 - Intel fanouts [618](#)
- syn_use_carry_chain attribute
 - using [709](#)
- syn_useioff
 - DSP block inference, Intel [591](#)
- syn_useioff attribute
 - packing registers (Intel) [643](#)
 - packing registers (Xilinx) [897](#)
- synchronous sets/resets
 - inferring registers with (Intel) [556](#)
 - Xilinx [858](#)
- SYNCore
 - adder/subtractor [418](#)
 - adder/subtractor parameters [427](#)
 - adders [419](#)
 - byte-enable RAM compiler
 - byte-enable RAM compiler
 - SYNCore** [387](#)
 - byte-enable RAM parameters [395](#)
 - counter compiler [442](#)

- counter parameters 449
 counters 443
 FIFO compiler 334, 335
 FIFO parameters 340
 RAM compiler
 RAM compiler
SYNCORE 365
 RAM parameters 373
 RAMs 365
 RAMs, byte-enable 388
 RAMs, dual-port parameters 371
 RAMs, single-port parameters 370
 ROM compiler 403
 ROM parameters 410
 ROMs 405
 ROMs, parameters 409
 subtractors 419
- SYNCORE** adder/subtractor
 adders 430
 dynamic adder/subtractor 436
 functional description 418
 subtractors 433
- SYNCORE** FIFOs
 definition 334
 parameter definitions 353
 port list 350
 read operations 350
 status flags 355
 write operations 349
- SYNCORE** ROMs
 clock latency 417
 dual-port read 415
 parameter list 416
 single-port read 414
- Synopsys customer support, contacting 32
- Synopsys FPGA Synthesis Tools
 overview 20
- Synopsys standard sdc file. *See* sdc files, difference between legacy and Synopsys standard
- Synplify Premier/DP
 Project view 34
- Synplify Premier/DP tools
 user interface 26
- Synplify Pro tool
 Project view 34
 user interface 26
- Synplify tool
 Project view 34
 user interface 26
 synplify.vhd 541, 950
 synthesis
 log file (.srr) 199
 watching selected information 54
 synthesis software
 gui 26
 system clock 208
 SystemVerilog keywords
 context help 65
- ## T
- ta file (customized timing report) 197
- Tcl commands
 constraint files 151
 pasting 58
- Tcl Script window
 Output Windows 62
- Tcl shell command
 sdc2fdc 154
- Tcl window
 popup menu commands 58
 popup menus 58
- technologies
 CPLD (Xilinx) 939
 Spartan (Xilinx) 933
 Virtex (Xilinx) 933
- Technology view 100
 displaying 79
 file (.srm) 195, 197
- Text Editor
 features 64
 indenting a block of text 64
 opening 63
 selecting text column 64
 view 62
- text editor
 completing keywords 64
- Text Editor view 62
- time borrowing 527
- timing analysis of critical paths (HDL Analyst tool) 139
- timing analyst

cross-clock paths 211
timing annotated properties (.tap) 197
timing constraints
 See also FPGA timing constraints
 See constraints
timing failures, definition 141
timing report 205
 clock relationships 211
 customized (.ta file) 197
 file (.ta) 197
 header 206
 interface information 212
 performance summary 206
Timing Report View
 accessing from Project Status View 48
timing reports
 asynchronous clocks 213
 log file (.srr) 203
title bar information, HDL Analyst tool 122
toolbars 76
 FSM 80, 81
 moving and docking 76
transparent hierarchical instances 112
lower-level logic on multiple sheets 124
operations resulting in 132
pins and pin names 115
trees of objects, Hierarchy Browser 102
trees, browser, collapsing and expanding 102
TriMatrix memory 559
true dual-port block RAM 856
true dual-port RAM
 See also multi-port RAMs

U

UCF constraints
 supported constraints 162
 translation examples 164
UINISIM library
 simulation 950
unfiltered schematic, compared with filtered 105
unfiltering schematic 134
unisim libraries

Virtex 2 with ISE 11 880
UNISIM library 880
user interface
 Synplify Premier/DP tools 26
 Synplify Pro tool 26
 Synplify tool 26
user interface, overview 33
using the mouse 71
utilities
 lib2syn 189
 sdc2fdc 154

V

v file 184, 185
vendor technologies
 Achronix 471
 Intel 547
 Lattice 665
 Microsemi 759
vendor-specific netlists 473, 667, 761, 835
Verilog
 clock DLLs 909
 Forward Annotation of Initial Values 316
 generic technology library 188
 initial value data file 313
 initial values for RAMs 310
 Intel PLLs 640
 macro library (Xilinx) 880
 Microsemi ACTgen macros 826
 netlist file 198
 RLOCs 913
 ROM inference 327
 source files (.v) 184, 185
 structural netlist file (.vm) 198
 supported standards 186

Verilog 2001 186
Verilog 95 186
Verilog macro libraries
 Lattice 703
 Microsemi 825
Verilog source file (.v) 185
vhd file 185
vhd source file 185
VHDL

clock DLLs 909
 Intel PLLs 640
 libraries
 attributes, supplied with synthesis tool 186
 macro libraries, Microsemi 825
 macro library (Xilinx) 880
 RLOCs 913
 source files (.vhd) 185
 structural netlist file (.vhm) 198
 supported standards 185

VHDL library
 specifying non-default (Intel) 552

VHDL macro libraries
 Lattice 704

VHDL source file (.vhd) 185

vhm file 198

views 53, 458
 FSM 103
 Project 34
 removing 76
 RTL 98
 Technology 100

Virtex
 clock buffers 908
 I/O buffers 911
 netlist 835
 PCI core 886

vm file 198

vqm files
 naming requirements 658

W

Watch Window. See Log Watch window

What's Cool 67

What's New 67

wide adder/subtractor 877

wide adders/subtractors
 example 885
 inferring 883
 prerequisites for inference 884

window
 Project 34

windows 53, 458
 closing 86
 log watch 54

removing 76

X

xc_clockbuftype attribute
 specifying 908

xc_fast attribute
 for critical paths 879

xc_loc attribute
 assigning locations in SCOPE 901

xc_map attribute
 relative location 912

xc_padtype attribute
 specifying I/Os 911

xc_rloc attribute
 specifying relative location 913

xc_uset attribute
 grouping instances for relative placement 913
 using to group instances 913

xflow and xtcish 947

Xilinx
 asymmetric block RAM 855
 attributes 951
 black boxes 866, 869
 block RAMs 856
 byte-enable RAM 302
 byte-enable RAM inference 302
 clock buffers 908
 clock skew 921
 compile point timing data 920
 compile points 948
 CoreGen 886
 CPLD device options 935
 CPLD options 937
 CPLD project command 939
 CPLD technologies 935, 939
 DDR register inference 860
 design guidelines 879
 directives 951
 distributed RAM 267
 distributed RAM inference 264
 dynamic SRL tables 864
 fanout limits 916
 gated clocks 919
 generated clocks 919
 GSR 882
 I/O buffers 911

I/O insertion, manual [905](#)
I/O locations [868](#), [900](#)
inferring dynamic SRL [864](#)
INIT property [890](#)
INIT property, VHDL [893](#)
instantiating I/O pads [867](#)
IP cores [886](#)
macro libraries [880](#)
macros [880](#)
mapping latches [866](#)
multi-port RAMs [856](#)
packing IOBs [867](#)
packing registers [897](#)
pipelining [918](#)
post-synthesis simulation [950](#)
register balancing [918](#)
register inference [858](#)
relative locations attributes [877](#)
resource usage report [941](#)
retiming [918](#)
SLR [837](#)
Spartan device options [925](#)
Spartan options [928](#)
Spartan project command [935](#)
Spartan technologies [933](#)
specifying pin location [868](#), [900](#)
startup blocks [882](#)
synthesis design flows [838](#)
technologies [833](#)
tips for optimizing [879](#)
true dual-port RAM. *See* multi-port
RAMs
updating compile points [920](#)
using BUFR [907](#)
Virtex device options [925](#)
Virtex options [928](#)
Virtex project command [935](#)
Virtex technologies [933](#)
XC4000 project command [935](#)
Xilinx differential I/O buffer inference
[867](#), [906](#)
Xilinx SSI technology [837](#)
xnf file (Xilinx) [834](#)
xtclsh executable [947](#)

Z

zoom
 using the mouse wheel and Ctrl key [75](#)

