Verification Continuum™

Synopsys Synplify Pro for Lattice

Reference Manual

November 2020

SYNOPSYS®

Copyright Notice and Proprietary Information

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

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

November 2020

Contents

Chapter 1: Product Overview

Overview of the Synthesis Tools 14
- Synplify Pro Features 14
- BEST Algorithms 15
- Graphic User Interface 15
- Projects and Implementations 18

Starting the Synthesis Tool 19

Logic Synthesis Overview 20
- Synthesizing Your Design 22

Getting Help 24

Chapter 2: User Interface Overview

The Project View 26
- Project Management View 28

The Project Results View 31
- Project Status Tab 31
- Implementation Directory 34
- Process View 36

Other Windows and Views 39
- Dockable GUI Entities 40
- Watch Window 40
- Tcl Script and Messages Windows 43
- Tcl Script Window 44
- Message Viewer 44
- Output Windows (Tcl Script and Watch Windows) 48
- RTL View 49
- Technology View 50
- Hierarchy Browser 52
- FSM Viewer Window 54
- Text Editor View 56

Context Help Editor Window 58
Interactive Attribute Examples 60

FSM Compiler 62
When to Use FSM Compiler 63
Where to Use FSM Compiler (Global and Local Use) 63

Using the Mouse 64
Mouse Operation Terminology 64
Using Mouse Strokes 65
Using the Mouse Buttons 66
Using the Mouse Wheel 68

Toolbars 69
Project Toolbar 69
Analyst Toolbar 71
Text Editor Toolbar 73
FSM Viewer Toolbar 74
Tools Toolbar 75

Keyboard Shortcuts 76

Buttons and Options 84

Chapter 3: HDL Analyst Tool

HDL Analyst Views and Commands 88
Filtered and Unfiltered Schematic Views 88
Accessing HDL Analyst Commands 89

Schematic Objects and Their Display 90
Object Information 90
Sheet Connectors 91
Primitive and Hierarchical Instances 92
Transparent and Opaque Display of Hierarchical Instances 93
Hidden Hierarchical Instances 95
Schematic Display 95

Basic Operations on Schematic Objects 99
Finding Schematic Objects 99
Selecting and Unselecting Schematic Objects 100
Crossprobing Objects 102
Dragging and Dropping Objects 103

Multiple-sheet Schematics 105
Controlling the Amount of Logic on a Sheet 105
Navigating Among Schematic Sheets 105

Multiple Sheets for Transparent Instance Details 107

Exploring Design Hierarchy 108
Pushing and Popping Hierarchical Levels 108
Navigating With a Hierarchy Browser 111
Looking Inside Hierarchical Instances 113

Filtering and Flattening Schematics 115
Commands That Result in Filtered Schematics 115
Combined Filtering Operations 116
Returning to The Unfiltered Schematic 117
Commands That Flatten Schematics 117
Selective Flattening 118
Filtering Compared to Flattening 120

Timing Information and Critical Paths 121
Timing Reports 121
Critical Paths and the Slack Margin Parameter 122
Examining Critical Path Schematics 124

Chapter 4: Constraints

Constraint Types 126

Constraint Files 127

Timing Constraints 129

FDC Constraints 132

Methods for Creating Constraints 133

Constraint Translation 135
sdc2fdc Conversion 135

Constraint Checking 140

Database Object Search 142

Forward Annotation 143

Auto Constraints 143

Chapter 5: Input and Result Files

Input Files 146
HDL Sourc e Files. 148

Libraries 150
The Generic Technology Library 151

ASIC Library Files 151
Open Verification Library 152

Output Files 154

Log File 160

Timing Reports 165
Timing Report Header 166
Performance Summary 166
Clock Relationships 168
Interface Information 170
Detailed Clock Report 170
A synchronous Clock Report 172

Constraint Checking Report 174

Gated Clock Conversion Report 181

Chapter 6: RAM and ROM Inference

Guidelines and Support for RAM Inference 184

Automatic RAM Inference 185
Block RAM 185
RAM Attributes 186

Block RAM Inference 189
Block RAM Examples 195

Initial Values for RAMs 208
Example 1: RAM Initialization 209
Example 2: Cross-Module Referencing for RAM Initialization 209
Initialization Data File 212
Forward Annotation of Initial Values 214

RAM Instantiation with SYNCORE 214

ROM Inference 215

Chapter 7: IP Tool

SYNCore FIFO Compiler 222
Synchronous FIFOs 222
FIFO Read and Write Operations 223
FIFO Ports 225
FIFO Parameters 227
FIFO Status Flags 229
FIFO Programmable Flags 232

SYNCore RAM Compiler 240
Single-Port Memories 240
Dual-Port Memories 242
Read/Write Timing Sequences 247

SYNCore Byte-Enable RAM Compiler 250
Functional Overview 250
Read/Write Timing Sequences 251
Parameter List 254

SYNCore ROM Compiler 256
Functional Overview 256
Single-Port Read Operation 257
Dual-Port Read Operation 258
Parameter List 259
Clock Latency 260

SYNCore Adder/Subtractor Compiler 262
Functional Description 262
Adder 263
Subtractor 266
Dynamic Adder/Subtractor 269

SYNCore Counter Compiler 274
Functional Overview 274
UP Counter Operation 275
Down Counter Operation 275
Dynamic Counter Operation 276

Chapter 8: Scripts

synhooks File Syntax 280

Tcl Script Examples 282
Using Target Technologies 282
Different Clock Frequency Goals 282
Setting Options and Timing Constraints 283

Appendix A: Designing with Lattice

Lattice Technology Support 288
Supported Device Families 288

IP Encryption Flow for Lattice 289

Lattice Features 290
RAM Support 290

DSP Support . . . 291
Inference Support for SB_MAC16 Blocks . . . 292
Lattice Oscillator Timing Propagation Support . . . 297
Block RAM Support . . . 298
SYNCORE RAMs . . . 309
Macros and Black Boxes . . . 309
Programmable I/O Cell Latches . . . 309
Initial Value Support for Registers . . . 310
Hierarchical Attribute Support for Module Generation . . . 311
Selective Clock Enable Inference . . . 311
Shift Chain Inference . . . 312
Lattice iCE40 and iCE40LM Components . . . 313
Forward Annotation . . . 322
Supported Timing Constraints . . . 323
Synthesis Reports . . . 323

Lattice Constraints, Attributes, and Options . . . 324
Constraints and Attributes . . . 324
Disable I/O Insertion Globally or Port by Port . . . 325
I/O Buffer Support . . . 325

Device Mapping Options . . . 327
Fanout Guide . . . 327
Disable I/O Insertion . . . 327
GSR Resource . . . 328
Compile Point Remapping in Lattice Designs . . . 328
Write Declared Clocks Only . . . 329

Design Options . . . 331
FSM Compiler . . . 331
Resource Sharing . . . 331
Pipelining . . . 331
Retiming . . . 332
Gated and Generated Clocks . . . 332

Implementation Results . . . 333

LatticeECP/EC and Later Devices . . . 334
Device Mapping Options . . . 334
Attributes . . . 334
set_option Tcl Command . . . 335

LatticeSC/SCM and LatticeXP2/XP Devices . . . 338
Device Mapping Options . . . 338
Attributes . . . 338
set_option Tcl Command . . . 339

Lattice MachXO and Platform Devices 341
    Device Mapping Options 341
    Attributes 342
    set_option Tcl Command 342

Lattice iCE40 Devices 345
    Device Mapping Options 345
    set_option Tcl Command 346

set_option Command for Lattice Architectures 348

Lattice Attribute and Directive Summary 352

CHAPTER 1

Product Overview

This document is part of a set that includes reference and procedural information for the Synopsys® FPGA synthesis tools. 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 tools.

- Overview of the Synthesis Tools, on page 14
- Starting the Synthesis Tool, on page 19
- Logic Synthesis Overview, on page 20
- Getting Help, on page 24

Overview of the Synthesis Tools

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

- Synplify Pro Features, on page 14
- BEST Algorithms, on page 15
- Graphic User Interface, on page 15
- Projects and Implementations, on page 18

Synplify Pro Features

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

- The HDL Analyst® RTL analysis and debugging environment, a graphical tool for analysis and crossprobing. See *RTL View*, on page 49, *Technology View*, on page 50, and *Analyzing With the Standard HDL Analyst Tool*, on page 352 in the *User Guide*.
- The Text Editor window, with a language-sensitive editor for writing and editing HDL code. See *Text Editor View*, on page 56.
- The SCOPE® (Synthesis Constraint Optimization Environment®) tool, which provides a spreadsheet-like interface for managing timing constraints and design attributes. See SCOPE User Interface, on page 154.
- FSM Compiler, a symbolic compiler that performs advanced finite state machine (FSM) optimizations. See *FSM Compiler*, on page 62.
- The FSM Viewer, for viewing state transitions in detail. See *FSM Viewer Window*, on page 54.
- The Tcl window, a command line interface for running TCL scripts. See *Tcl Script Window*, on page 44.
- 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 26).

- Place-and-Route implementation(s) to automatically run placement and routing after synthesis. You can run place-and-route from within the tool or in batch mode. This feature is supported for the iCE technologies (see *Running P&R Automatically after Synthesis,* on page 652 in the *User Guide*).
- Certain optimizations, like pipelining and retiming are only available with these tools.
- Advanced analysis features like crossprobing.

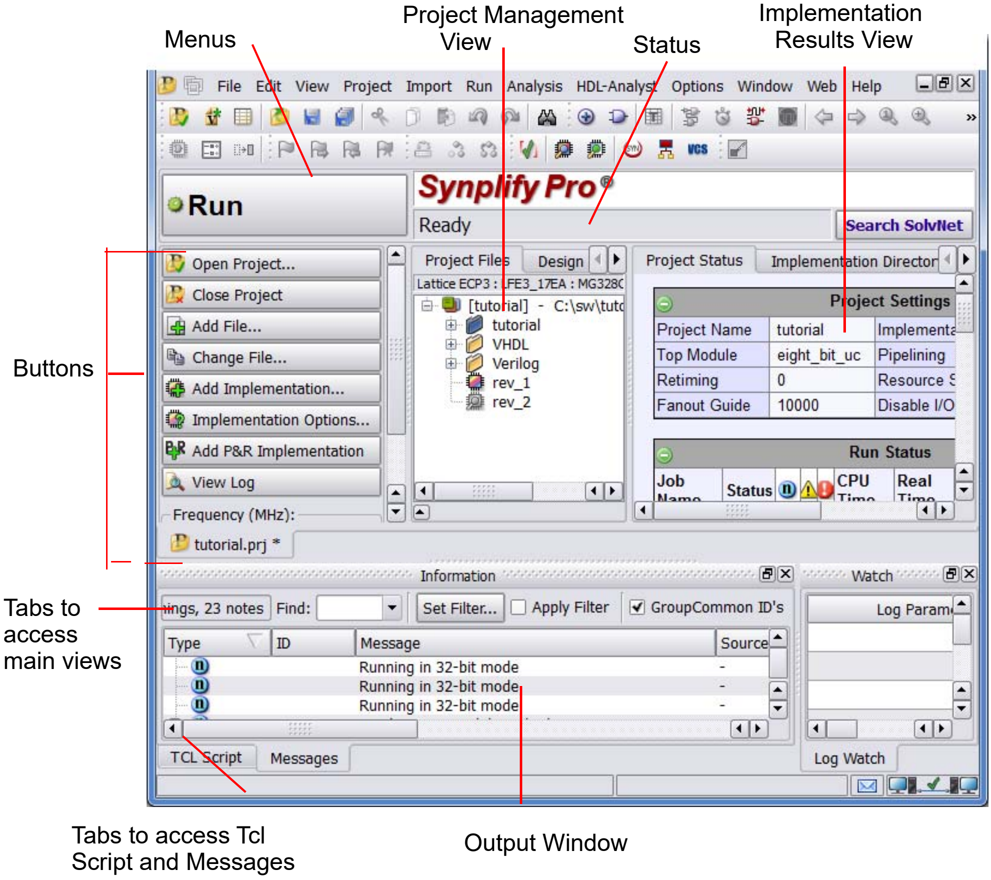
BEST Algorithms

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

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

Graphic User Interface

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



Project Settings

Project Name	tutorial	Implementa
Top Module	eight_bit_uc	Pipelining
Retiming	0	Resource S
Fanout Guide	10000	Disable I/O

Run Status

Job Name	Status	CPU Time	Real Time

Type	ID	Message	Source
ⓝ		Running in 32-bit mode	-
ⓝ		Running in 32-bit mode	-
ⓝ		Running in 32-bit mode	-

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 26
RTL view	*RTL View* , on page 49
Technology view	*Technology View* , on page 50
Text Editor view	*Text Editor View* , on page 56
FSM Viewer window	*FSM Viewer Window* , on page 54
Tcl window	*Tcl Script Window* , on page 44
Watch Window	*Watch Window* , on page 40
SCOPE spreadsheet	SCOPE User Interface, on page 154
Other views and windows	The Project View, on page 26
Menu commands and their dialog boxes	Chapter 2, *User Interface Overview*
Toolbars	*Toolbars* , on page 69
Buttons	*Buttons and Options* , on page 84
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 489, for more information.

Projects and Implementations

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

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

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

Starting the Synthesis Tool

Before you can start the synthesis tool, you must install it and set up the software license appropriately. You can then start the tool interactively or in batch mode. How you start the tool depends on your environment. For details, see the installation instructions for the tool.

Starting the Synthesis Tool in Interactive Mode

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

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

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

For information about using the synthesis tool in batch mode, see *Starting the Tool in Batch Mode,* on page 20.

Starting the Tool in Batch Mode

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

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

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

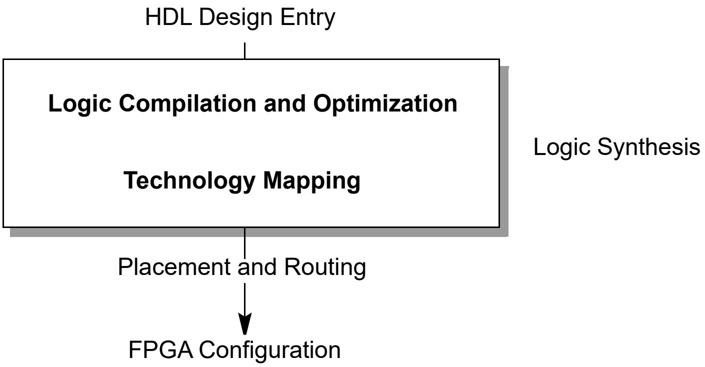
- synplify_pro

For complete syntax details, refer to *synplify_pro*, on page 142 in the *Command Reference*.

Logic Synthesis Overview

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

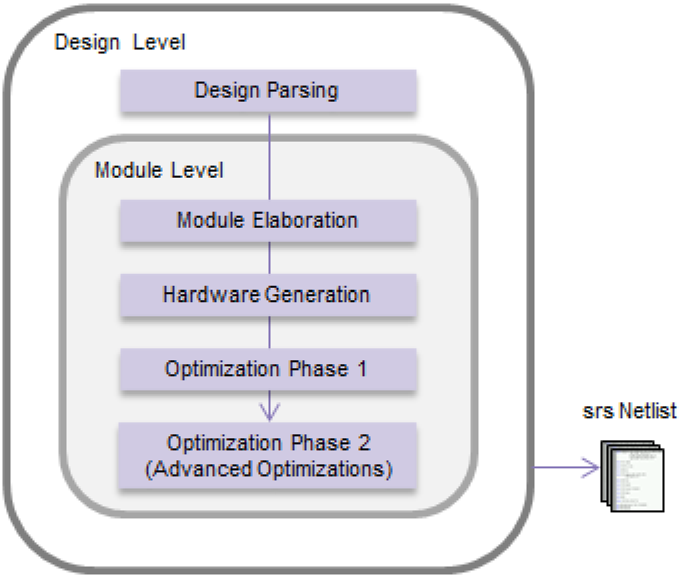
- Logic compilation (HDL language synthesis) and optimization
- Technology mapping



Logic Compilation

The synthesis tool first compiles input HDL source code, which describes the design at a high level of abstraction, to known structural elements. Next, it optimizes the design in two phases, making it as small as possible to improve circuit performance. These optimizations are technology independent. The final result is an srs database, which can be graphically represented in the RTL schematic view.

The following figure summarizes the stages of the standard compiler flow:



You can also run the compiler incrementally.

Technology Mapping

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

Synthesizing Your Design

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

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

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

   See SCOPE Tabs, on page 155, for details on the SCOPE spreadsheet.
4. Choose Project->Implementation Options, then define the following:
   - Target architecture and technology specifications
   - Optimization options and design constraints
   - Outputs

   For an initial run, use the default options settings for the technology, and no timing goal (Frequency = 0 MHz).
5. Synthesize the design by clicking the Run button.

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

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

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

Getting Help

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

Finding Information

For help with...	Refer to the...
How to...	*User Guide* and various application notes available on the Synopsys support website
Flow information	*User Guide* and various application notes available on the Synopsys Support website
FPGA Implementation Tools	Synopsys Web Page (Web->FPGA Implementation Tools menu command from within the software)
Synthesis features	*User Guide* and *Reference Manual*
Language and syntax	*Language Support Reference Manual*
Attributes and directives	*Attribute Reference Manual*
Tcl language	Online help (Help->Tcl Help)
Synthesis Tcl commands	*Command Reference Manual* or type help followed by the command name in the Tcl window
Using tool-specific features and attributes	*User Guide*
Error and warning messages	Click the message ID code

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 26
- The Project Results View, on page 31
- Other Windows and Views, on page 39
- FSM Compiler, on page 62
- Using the Mouse, on page 64
- Toolbars, on page 69
- Keyboard Shortcuts, on page 76
- Buttons and Options, on page 84

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. See the following for an overview:

- Multiple Pane Project View, on page 26
- Project Management View, on page 28

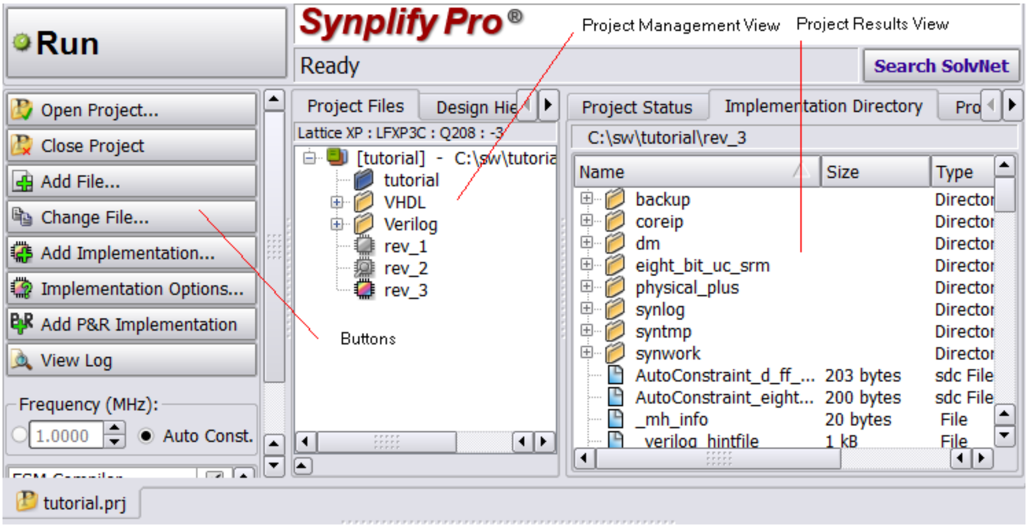
Multiple Pane Project View

The Project Management view is on the left side of the window, and is used to create or open projects, create new implementations, set device options, and initiate design synthesis. The Project Results view is on the right.

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 28
- The Project Results View



Lattice XP : LFXP3C : Q208 : -3

Frequency (MHz): ○ 1.0000 ● Auto Const.

C:\sw\tutorial\rev_3

Name	Size	Type
backup		Director
coreip		Director
dm		Director
eight_bit_uc_srm		Director
physical_plus		Director
synlog		Director
syntmp		Director
synwork		Director
AutoConstraint_d_ff_...	203 bytes	sdc File
AutoConstraint_eight...	200 bytes	sdc File
_mh_info	20 bytes	File
_verilog_hintfile	1 kB	File

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.
Buttons and options	Allow immediate access to some of the more common commands. See *Buttons and Options* , on page 84 for details.
Hierarchical Project Management view	Lists the projects and implementations, and their associated HDL source files and constraint files. There are two tabs with different views to facilitate working with hierarchical designs. • Project Files Tab • Design Hierarchy Tab
Implementation Results view	Lists the result of the synthesis runs for the implementations of your design. You can only view one set of implementation results at a time. Click an implementation in the Project view to make it active and view its result files. The Project Results view includes the following: • Project Status Tab—provides an overview of the project settings and at-a-glance summary of synthesis messages and reports. • Implementation Directory—lists the names and types of the result files, and the dates they were last modified. • Process View—gives you instant visibility to the synthesis and place-and-route job flows. See *The Project Results View* , on page 31 for more information.

The Project Management view is on the left side of the window, and is used to create or open projects, create new implementations, set device options, and initiate design synthesis. The Project Results view is on the right.

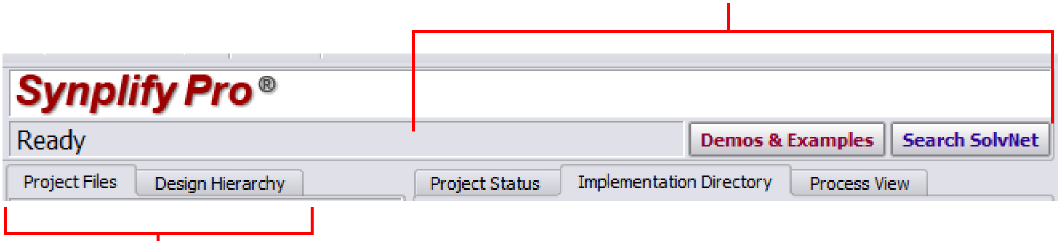
The Project view has the following main parts:

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
Buttons and options	Allow immediate access to some of the more common commands. See *Buttons and Options* , on page 84 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 18 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 465).

Project Management View

Project Results View



Project Management Views

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 32 in the User Guide*.

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

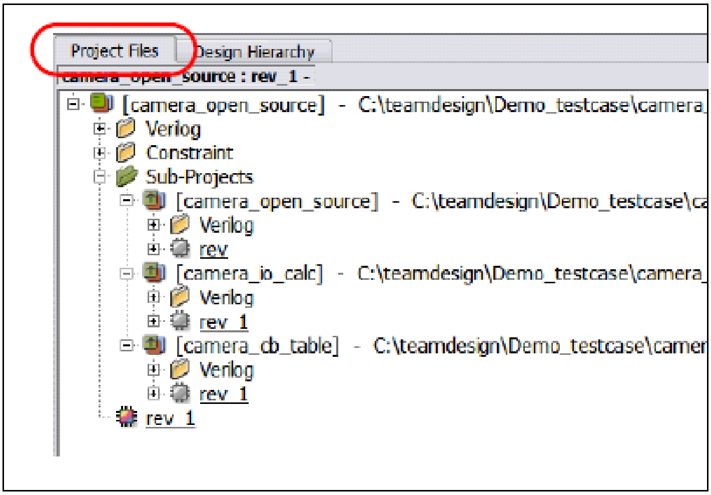
- Project Files Tab
- Design Hierarchy Tab

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 504*.

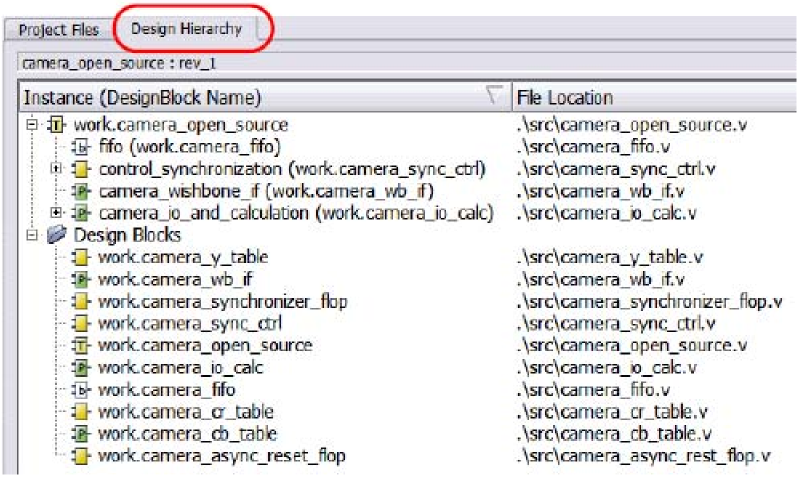
Project Files Tab

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

Design Hierarchy Tab


Project Files
Design Hierarchy
camera_open_source : rev_1 -
[camera_open_source] - C:\teamdesign\Demo_testcase\camera_
Verilog
Constraint
Sub-Projects
[camera_open_source] - C:\teamdesign\Demo_testcase\ca
Verilog
rev
[camera_io_calc] - C:\teamdesign\Demo_testcase\camera_
Verilog
rev_1
[camera_cb_table] - C:\teamdesign\Demo_testcase\camer
Verilog
rev_1
rev_1

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



camera_open_source : rev_1

Instance (DesignBlock Name)	File Location
work.camera_open_source	.\src\camera_open_source.v
fifo (work.camera_fifo)	.\src\camera_fifo.v
control_synchronization (work.camera_sync_ctrl)	.\src\camera_sync_ctrl.v
camera_wishbone_if (work.camera_wb_if)	.\src\camera_wb_if.v
camera_io_and_calculation (work.camera_io_calc)	.\src\camera_io_calc.v
Design Blocks	
work.camera_y_table	.\src\camera_y_table.v
work.camera_wb_if	.\src\camera_wb_if.v
work.camera_synchronizer_flop	.\src\camera_synchronizer_flop.v
work.camera_sync_ctrl	.\src\camera_sync_ctrl.v
work.camera_open_source	.\src\camera_open_source.v
work.camera_io_calc	.\src\camera_io_calc.v
work.camera_fifo	.\src\camera_fifo.v
work.camera_cr_table	.\src\camera_cr_table.v
work.camera_cb_table	.\src\camera_cb_table.v
work.camera_async_reset_flop	.\src\camera_async_rest_flop.v

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

Project Status Tab

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

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

- Project Settings
- Run Status
- Reports

For details about how to access synthesis results, see *Accessing Specific Reports Quickly*, on page 223.

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

Project Settings

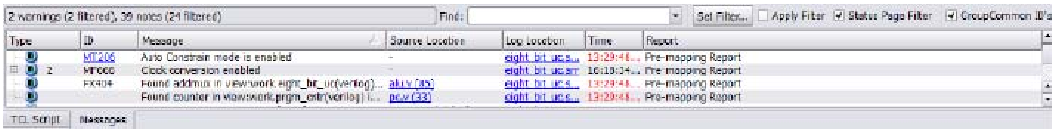
Project Settings is populated with the project settings from the run_options.txt file after a synthesis run. This section displays information, like the following:

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

Run Status

The Run Status table gets updated during and after a synthesis run. This section displays job status information for the compiler, premap job, mapper, and place-and-route runs, as needed. This section displays information 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 on this link, it takes you to the corresponding report in the log file.
- 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.



2 warnings (2 filtered), 39 notes (24 filtered)

Type	ID	Message	Source Location	Log Location	Time	Report
	MT206	Auto Constrain mode is enabled	-	eight_bit_uc.s...	13:29:48...	Pre-mapping Report
2	MF000	Clock conversion enabled		eight_bit_uc.srr	16:18:34...	Pre-mapping Report
	FX404	Found addmux in view:work.eight_bit_uc(verilog)...	alu.v (35)	eight_bit_uc.s...	13:29:48...	Pre-mapping Report
		Found counter in view:work.prgm_cntr(verilog) i...	pc.v (33)	eight_bit_uc.s...	13:29:48...	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:

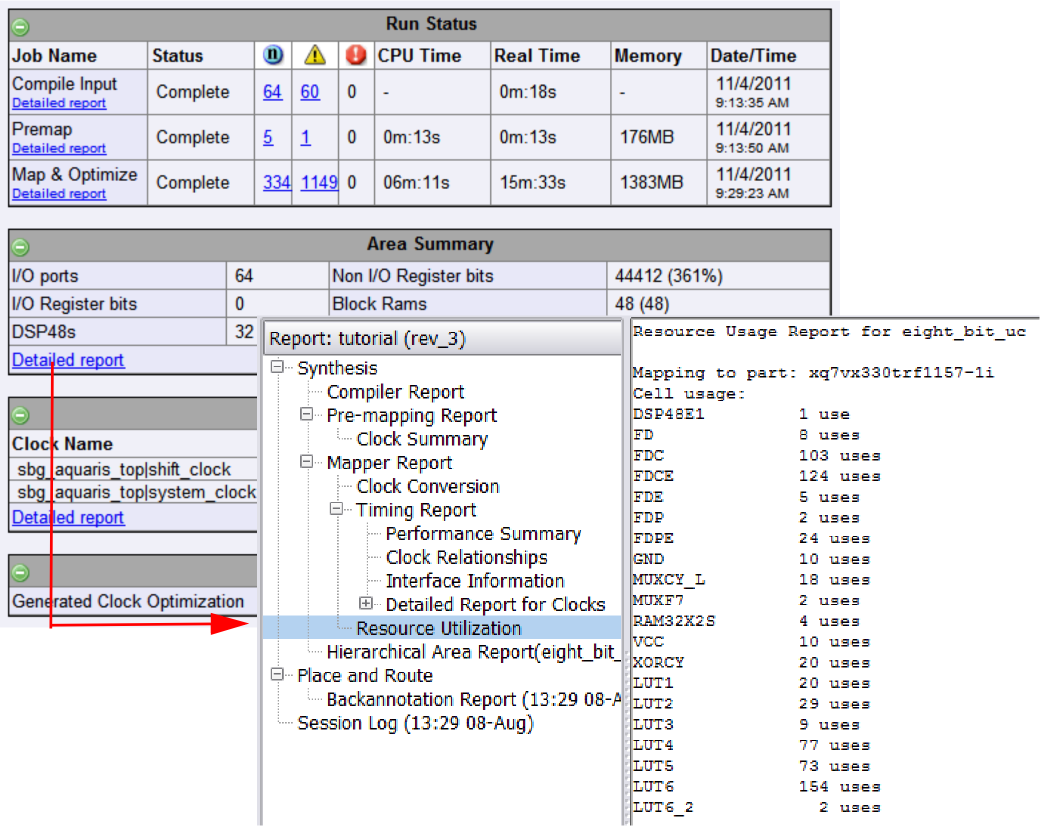
- Area Summary
- Compile Point Summary
- Optimization Summary

- 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	ⓘ	⚠	❗	CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	64	60	0	-	0m:18s	-	11/4/2011 9:13:35 AM
Premap Detailed report	Complete	5	1	0	0m:13s	0m:13s	176MB	11/4/2011 9:13:50 AM
Map & Optimize Detailed report	Complete	334	1149	0	06m:11s	15m:33s	1383MB	11/4/2011 9:29:23 AM

Area Summary

I/O ports	64	Non I/O Register bits	44412 (361%)
I/O Register bits	0	Block Rams	48 (48)
DSP48s	32		
Detailed report			

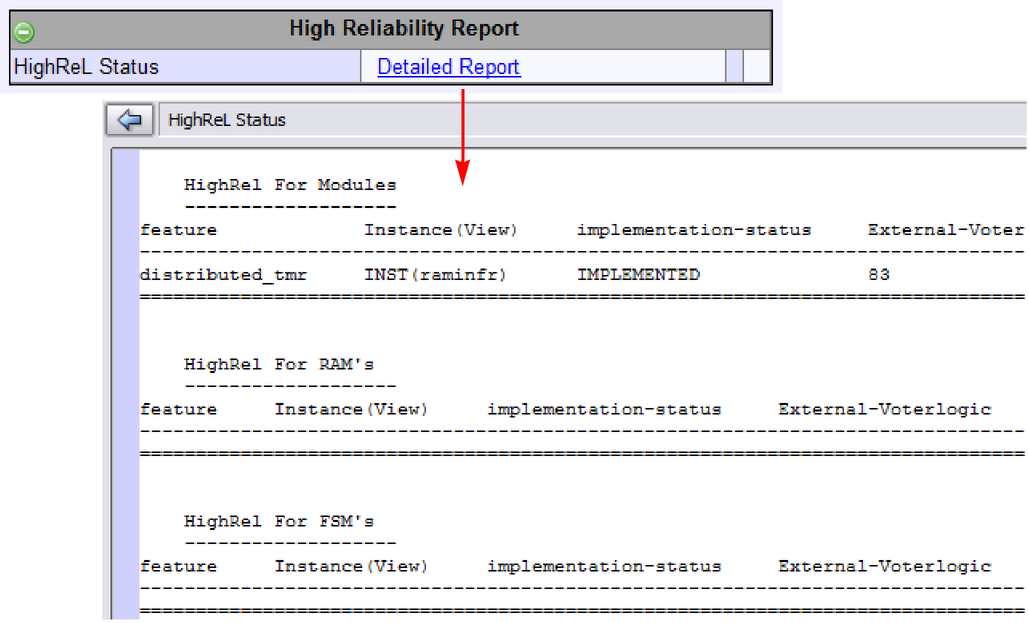
Clock Name
sbg_aquaris_top|shift_clock
sbg_aquaris_top|system_clock
Detailed report

Generated Clock Optimization

```
Resource Usage Report for eight_bit_uc

Mapping to part: xq7vx330trf1157-1i
Cell usage:
DSP48E1         1 use
FD              8 uses
FDC             103 uses
FDCE            124 uses
FDE             5 uses
FDP             2 uses
FDPE            24 uses
GND             10 uses
MUXCY_L         18 uses
MUXF7           2 uses
RAM32X2S        4 uses
VCC             10 uses
XORCY           20 uses
LUT1            20 uses
LUT2            29 uses
LUT3            9 uses
LUT4            77 uses
LUT5            73 uses
LUT6            154 uses
LUT6_2            2 uses
```

High Reliability Report

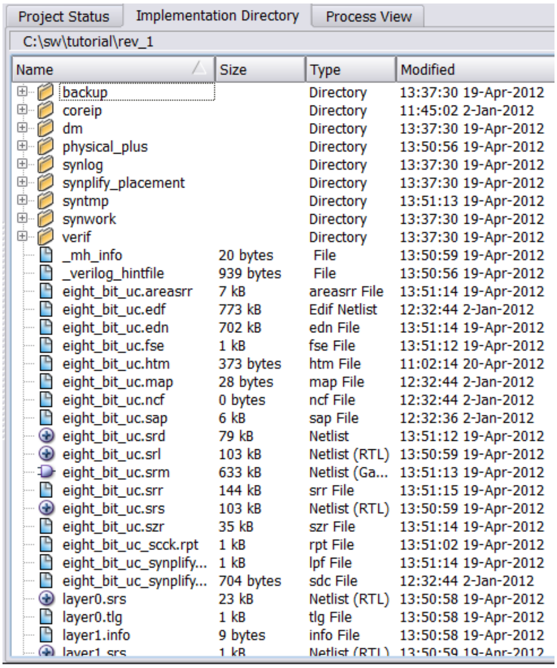
Click Detailed Report in the High Reliability Report section of the Project Status view to view the high reliability status report. The High Reliability Report is displayed only if one of the High Reliability features, such as Safe FSMs is being used.



```
    HighRel For Modules
    -------------------
feature             Instance(View)     implementation-status     External-Voter
-------------------------------------------------------------------------------
distributed_tmr     INST(raminfr)      IMPLEMENTED               83
===============================================================================


    HighRel For RAM's
    -------------------
feature     Instance(View)     implementation-status     External-Voterlogic
-------------------------------------------------------------------------------
===============================================================================


    HighRel For FSM's
    -------------------
feature     Instance(View)     implementation-status     External-Voterlogic
-------------------------------------------------------------------------------
===============================================================================
```

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.



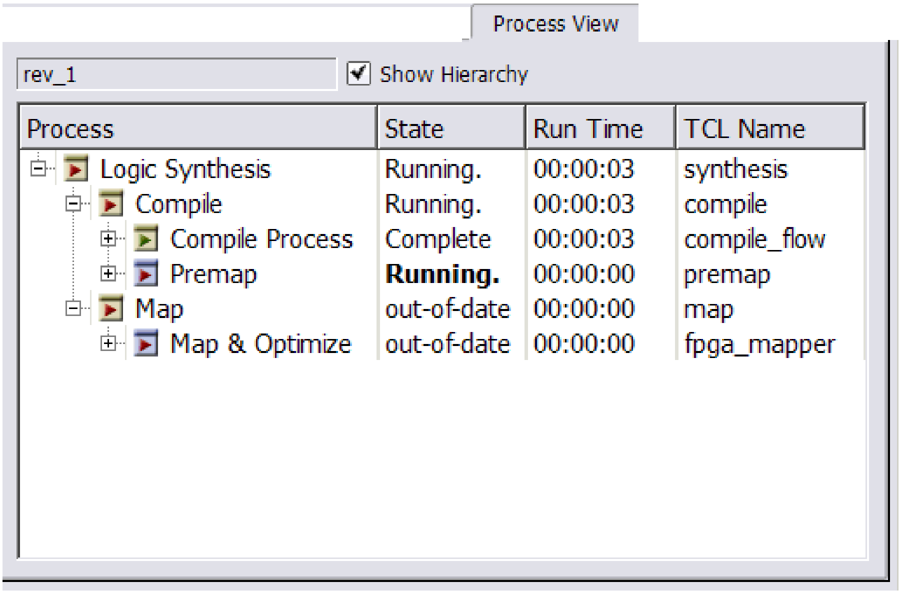
C:\sw\tutorial\rev_1

Name	Size	Type	Modified
backup		Directory	13:37:30 19-Apr-2012
coreip		Directory	11:45:02 2-Jan-2012
dm		Directory	13:37:30 19-Apr-2012
physical_plus		Directory	13:50:56 19-Apr-2012
synlog		Directory	13:37:30 19-Apr-2012
synplify_placement		Directory	13:37:30 19-Apr-2012
syntmp		Directory	13:51:13 19-Apr-2012
synwork		Directory	13:37:30 19-Apr-2012
verif		Directory	13:37:30 19-Apr-2012
_mh_info	20 bytes	File	13:50:59 19-Apr-2012
_verilog_hintfile	939 bytes	File	13:50:56 19-Apr-2012
eight_bit_uc.areasrr	7 kB	areasrr File	13:51:14 19-Apr-2012
eight_bit_uc.edf	773 kB	Edif Netlist	12:32:44 2-Jan-2012
eight_bit_uc.edn	702 kB	edn File	13:51:14 19-Apr-2012
eight_bit_uc.fse	1 kB	fse File	13:51:12 19-Apr-2012
eight_bit_uc.htm	373 bytes	htm File	11:02:14 20-Apr-2012
eight_bit_uc.map	28 bytes	map File	12:32:44 2-Jan-2012
eight_bit_uc.ncf	0 bytes	ncf File	12:32:44 2-Jan-2012
eight_bit_uc.sap	6 kB	sap File	12:32:36 2-Jan-2012
eight_bit_uc.srd	79 kB	Netlist	13:51:12 19-Apr-2012
eight_bit_uc.srl	103 kB	Netlist (RTL)	13:50:59 19-Apr-2012
eight_bit_uc.srm	633 kB	Netlist (Ga...	13:51:13 19-Apr-2012
eight_bit_uc.srr	144 kB	srr File	13:51:15 19-Apr-2012
eight_bit_uc.srs	103 kB	Netlist (RTL)	13:50:59 19-Apr-2012
eight_bit_uc.szr	35 kB	szr File	13:51:14 19-Apr-2012
eight_bit_uc_scck.rpt	1 kB	rpt File	13:51:02 19-Apr-2012
eight_bit_uc_synplify...	1 kB	lpf File	13:51:14 19-Apr-2012
eight_bit_uc_synplify...	704 bytes	sdc File	12:32:44 2-Jan-2012
layer0.srs	23 kB	Netlist (RTL)	13:50:58 19-Apr-2012
layer0.tlg	1 kB	tlg File	13:50:58 19-Apr-2012
layer1.info	9 bytes	info File	13:50:58 19-Apr-2012
layer1.srs	1 kB	Netlist (RTL)	13:50:59 19-Apr-2012

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.



rev_1 ☑ Show Hierarchy

Process	State	Run Time	TCL Name
Logic Synthesis	Running.	00:00:03	synthesis
Compile	Running.	00:00:03	compile
Compile Process	Complete	00:00:03	compile_flow
Premap	**Running.**	00:00:00	premap
Map	out-of-date	00:00:00	map
Map & Optimize	out-of-date	00:00:00	fpga_mapper

Process View Displays and Controls

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

Appropriate jobs of the process flow contains the following information:

- Job Input and Output Files
- Completion State

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

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

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

Hierarchical Job Flows

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

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

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

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

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

Other Windows and Views

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

- Dockable GUI Entities, on page 40
- Watch Window, on page 40
- Tcl Script and Messages Windows, on page 43
- Tcl Script Window, on page 44
- Message Viewer, on page 44
- Output Windows (Tcl Script and Watch Windows), on page 48
- RTL View, on page 49
- Technology View, on page 50
- Hierarchy Browser, on page 52
- FSM Viewer Window, on page 54
- Text Editor View, on page 56
- Context Help Editor Window, on page 58
- Interactive Attribute Examples, on page 60

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

Project view	The Project View, on page 26
SCOPE	SCOPE Tabs, on page 155

Dockable GUI Entities

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

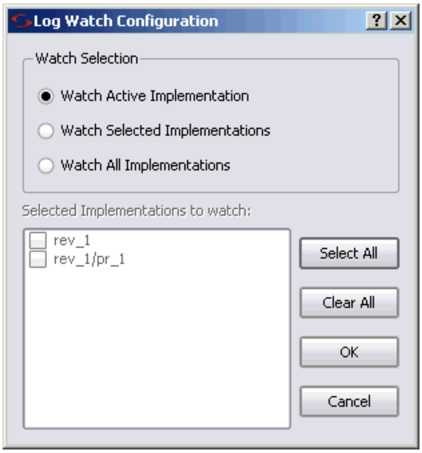
Watch Window

The Watch window displays selected information from the log file (see *Log File*, on page 160) 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.



● Watch Active Implementation
○ Watch Selected Implementations
○ Watch All Implementations

□ rev_1
□ rev_1/pr_1

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.

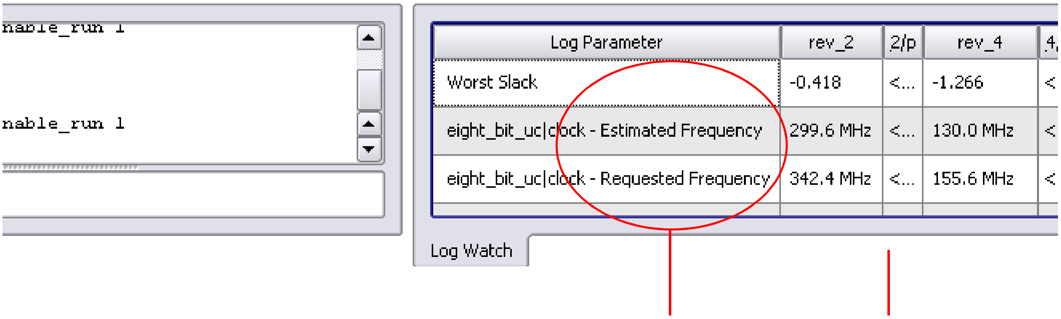
The Watch window has a special positioning popup menu that you access by right-clicking the window border. The following commands are in the menu:

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.

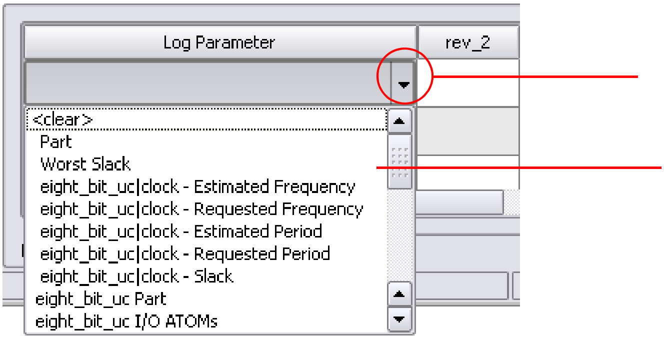
Using the Watch Window

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



Log Parameter	rev_2	2/p	rev_4	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 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:

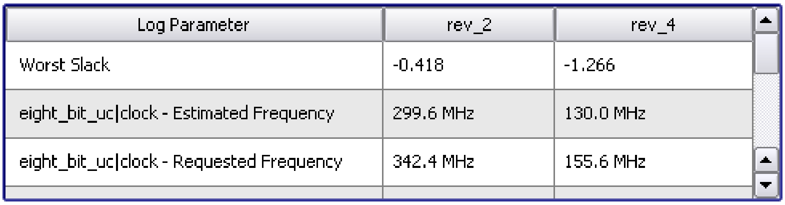

Log Parameter
rev_2
<clear>
Part
Worst Slack
eight_bit_uc|clock - Estimated Frequency
eight_bit_uc|clock - Requested Frequency
eight_bit_uc|clock - Estimated Period
eight_bit_uc|clock - Requested Period
eight_bit_uc|clock - Slack
eight_bit_uc Part
eight_bit_uc I/O ATOMs

Click pull-down arrow

to

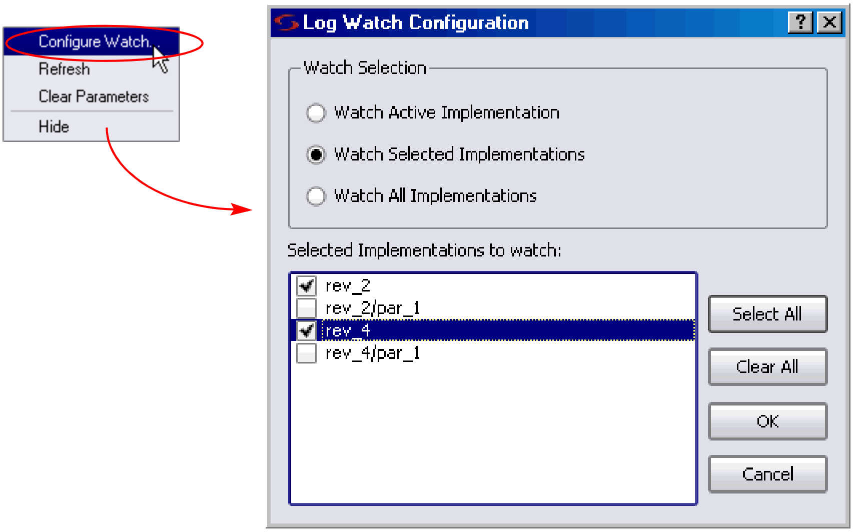
display list of choices

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.



Watch Selection

○ Watch Active Implementation

● Watch Selected Implementations

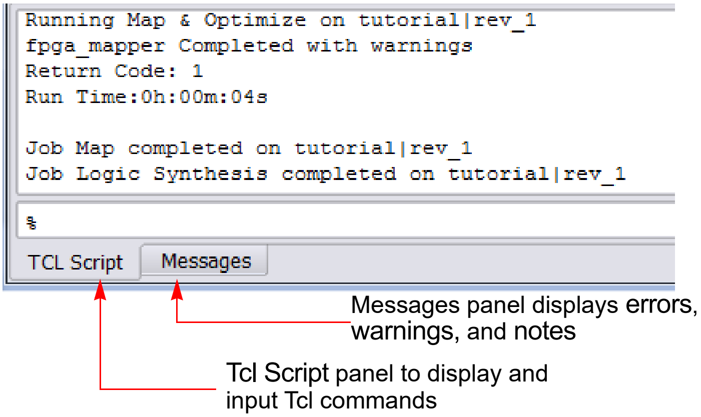
○ Watch All Implementations

Selected Implementations to watch:

☑ rev_2
□ rev_2/par_1
☑ rev_4
□ rev_4/par_1

Tcl Script and Messages Windows

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.



```
Running Map & Optimize on tutorial|rev_1
fpga_mapper Completed with warnings
Return Code: 1
Run Time:0h:00m:04s

Job Map completed on tutorial|rev_1
Job Logic Synthesis completed on tutorial|rev_1
```

```
%
```

Messages panel displays errors, warnings, and notes

Tcl Script panel to display and input Tcl commands

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 44 and *Message Viewer*, on page 44.

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.

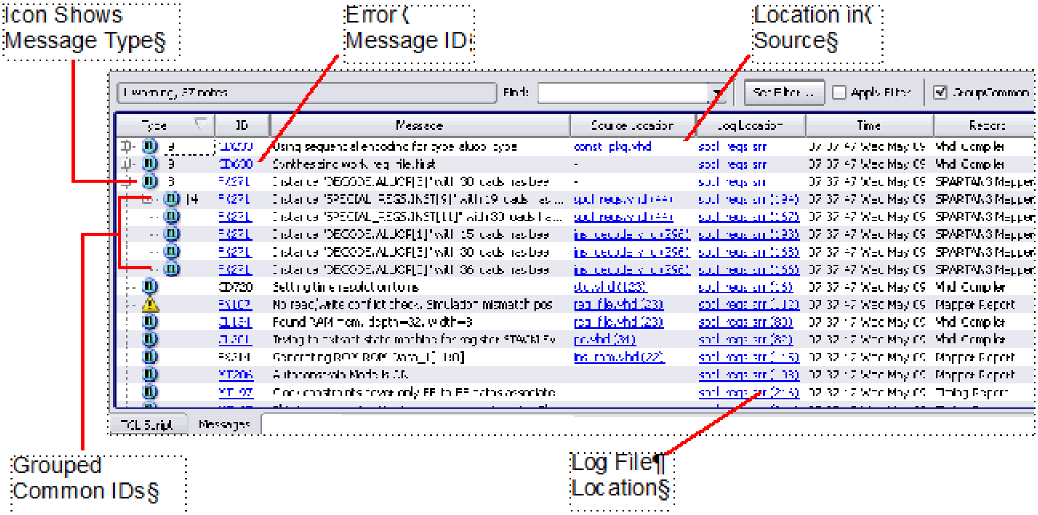
The Tcl script window also displays each command executed in the course of running the synthesis tool, regardless of whether it was initiated from a menu, button, or keyboard shortcut. Right-clicking inside the Tcl window displays a popup menu with the Copy, Paste, Hide, and Help commands.

See also

- *Generating a Job Script*, on page 573 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.



Type	ID	Message	Source Location	Log Location	Time	Report
9	CD233	Using sequential encoding for type aluop_type	const_pkg.vhd	spcl_regs.srr	07:37:47 Wed May 09	Vhdl Compiler
9	CD630	Synthesizing work.reg_file.first	-	spcl_regs.srr	07:37:47 Wed May 09	Vhdl Compiler
3	FX271	Instance "DECODE.ALUOP[3]" with 30 loads has bee	-	spcl_regs.srr	07:37:47 Wed May 09	SPARTAN3 Mapper
14	FX271	Instance "SPECIAL_REGS.INST[9]" with 19 loads has ...	spcl_regs.vhd(44)	spcl_regs.srr(194)	07:37:47 Wed May 09	SPARTAN3 Mapper
	FX271	Instance "SPECIAL_REGS.INST[11]" with 30 loads ha...	spcl_regs.vhd(44)	spcl_regs.srr(167)	07:37:47 Wed May 09	SPARTAN3 Mapper
	FX271	Instance "DECODE.ALUOP[1]" with 15 loads has bee	ins_decode.vhd(256)	spcl_regs.srr(193)	07:37:47 Wed May 09	SPARTAN3 Mapper
	FX271	Instance "DECODE.ALUOP[3]" with 30 loads has bee	ins_decode.vhd(256)	spcl_regs.srr(153)	07:37:47 Wed May 09	SPARTAN3 Mapper
	FX271	Instance "DECODE.ALUOP[2]" with 36 loads has bee	ins_decode.vhd(256)	spcl_regs.srr(155)	07:37:47 Wed May 09	SPARTAN3 Mapper
	CD720	Setting time resolution to ns	std.vhd(123)	spcl_regs.srr(15)	07:37:47 Wed May 09	Vhdl Compiler
⚠	FX107	No read/write conflict check. Simulation mismatch pos	reg_file.vhd(23)	spcl_regs.srr(113)	07:37:47 Wed May 09	Mapper Report
	CL134	Found RAM mem, depth=32, width=8	reg_file.vhd(23)	spcl_regs.srr(80)	07:37:47 Wed May 09	Vhdl Compiler
	CL201	Trying to extract state machine for register STACK_Pv	pc.vhd(31)	spcl_regs.srr(82)	07:37:47 Wed May 09	Vhdl Compiler
	FX214	Generating ROM ROM_data_1[10:0]	ins_rom.vhd(22)	spcl_regs.srr(115)	07:37:47 Wed May 09	Mapper Report
	MT206	Autoconstrain Mode is ON		spcl_regs.srr(138)	07:37:47 Wed May 09	Mapper Report
	MT197	Clock constraints cover only FF-to-FF paths associate		spcl_regs.srr(245)	07:37:47 Wed May 09	Timing Report

Interactive tasks in the Messages panel include:

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

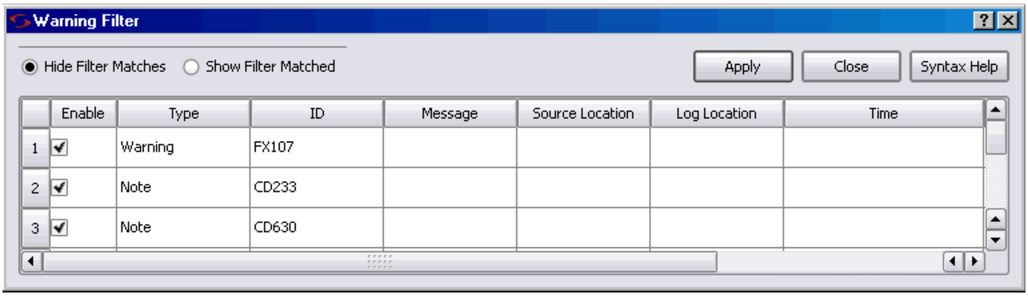
The following table describes the contents of the Messages panel. You can sort the messages by clicking the column headers. For further sorting, use Find and Filter. For details about using this window, see *Checking Results in the Message Viewer*, on page 234 in the *User Guide*.

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 47.
Apply Filter	Enable/disable the last saved filter.

Item	Description
Group Common ID's	Enable/disable grouping of repeated messages. Groups are indicated by a number next to the type icon. There are two types of groups: • The same warning or note ID appears in multiple source files indicated by a dash in the source files column. • Multiple warnings or notes in the same line of source code indicated by a bracketed number.
Type	The icons indicate the type of message: ❗ Error ⚠ Warning n Note a 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 234 in the *User Guide*.

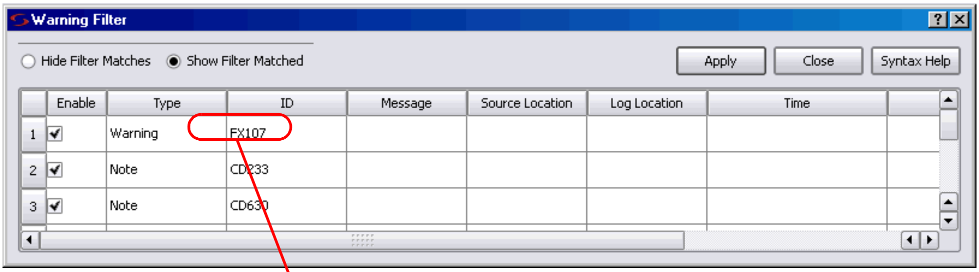


● Hide Filter Matches ○ Show Filter Matched

	Enable	Type	ID	Message	Source Location	Log Location	Time
1	☑	Warning	FX107				
2	☑	Note	CD233				
3	☑	Note	CD630				

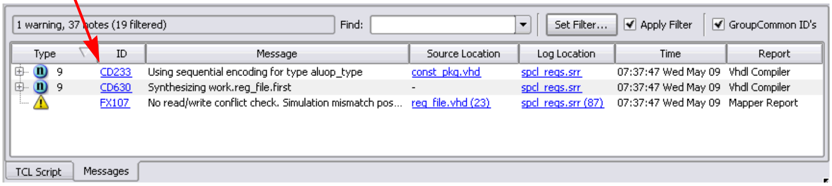
Item	Description
Hide Filter Matches	Hides matched criteria in the Messages Panel.
Show Filter Matches	Shows matched criteria in the Messages Panel.
Syntax Help	Gives quick syntax descriptions.
Apply	Applies the filter criteria to the Messages Panel report, without closing the window.
Type, ID, Message,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

The following is a filtering example.



	Enable	Type	ID	Message	Source Location	Log Location	Time
1	☑	Warning	FX107				
2	☑	Note	CD233				
3	☑	Note	CD630				

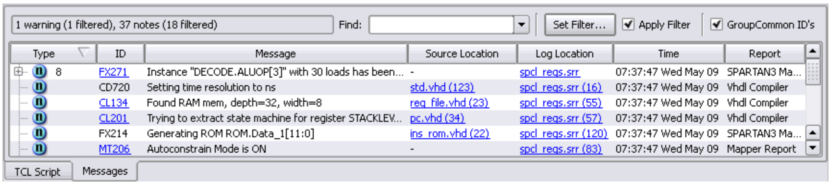
Show Filter Matches



1 warning, 37 notes (19 filtered)

Type		ID	Message	Source Location	Log Location	Time	Report
⊞ n	9	CD233	Using sequential encoding for type aluop_type	const_pkg.vhd	spcl_regs.srr	07:37:47 Wed May 09	Vhdl Compiler
⊞ n	9	CD630	Synthesizing work.reg_file.first	-	spcl_regs.srr	07:37:47 Wed May 09	Vhdl Compiler
⚠		FX107	No read/write conflict check. Simulation mismatch pos...	reg_file.vhd (23)	spcl_regs.srr (87)	07:37:47 Wed May 09	Mapper Report

Hide Filter Matches



1 warning (1 filtered), 37 notes (18 filtered)

Type		ID	Message	Source Location	Log Location	Time	Report
⊞ n	8	FX271	Instance "DECODE.ALUOP[3]" with 30 loads has been...	-	spcl_regs.srr	07:37:47 Wed May 09	SPARTAN3 Ma...
n		CD720	Setting time resolution to ns	std.vhd (123)	spcl_regs.srr (16)	07:37:47 Wed May 09	Vhdl Compiler
n		CL134	Found RAM mem, depth=32, width=8	reg_file.vhd (23)	spcl_regs.srr (55)	07:37:47 Wed May 09	Vhdl Compiler
n		CL201	Trying to extract state machine for register STACKLEV...	pc.vhd (34)	spcl_regs.srr (57)	07:37:47 Wed May 09	Vhdl Compiler
n		FX214	Generating ROM ROM.Data_1[11:0]	ins_rom.vhd (22)	spcl_regs.srr (120)	07:37:47 Wed May 09	SPARTAN3 Ma...
n		MT206	Autoconstrain Mode is ON	-	spcl_regs.srr (83)	07:37:47 Wed May 09	Mapper Report

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 40 and *Tcl Script and Messages Windows*, on page 43 for more information.

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 50.

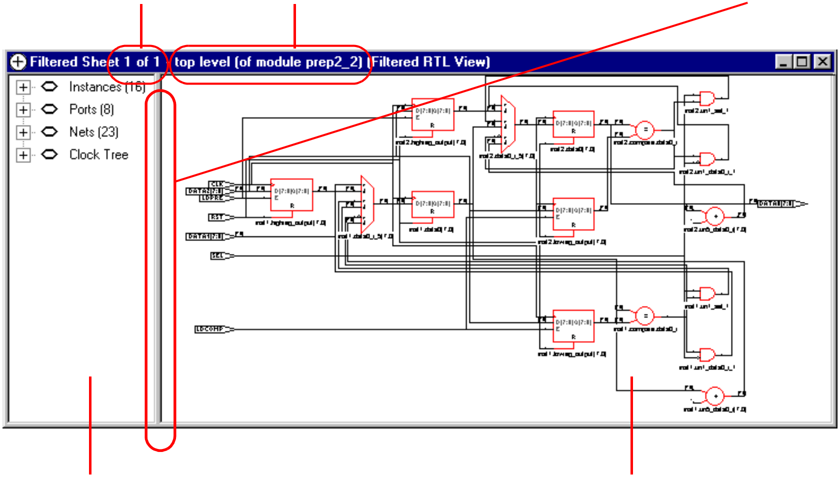
To display an RTL view, first compile or synthesize your design, then select HDL Analyst->RTL and choose Hierarchical View or Flattened View, or click the RTL icon (⊕).

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 52. 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.

Sheet # of total #

Current schematic level

Movable pane divider


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

Hierarchy Browser

Schematic

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:

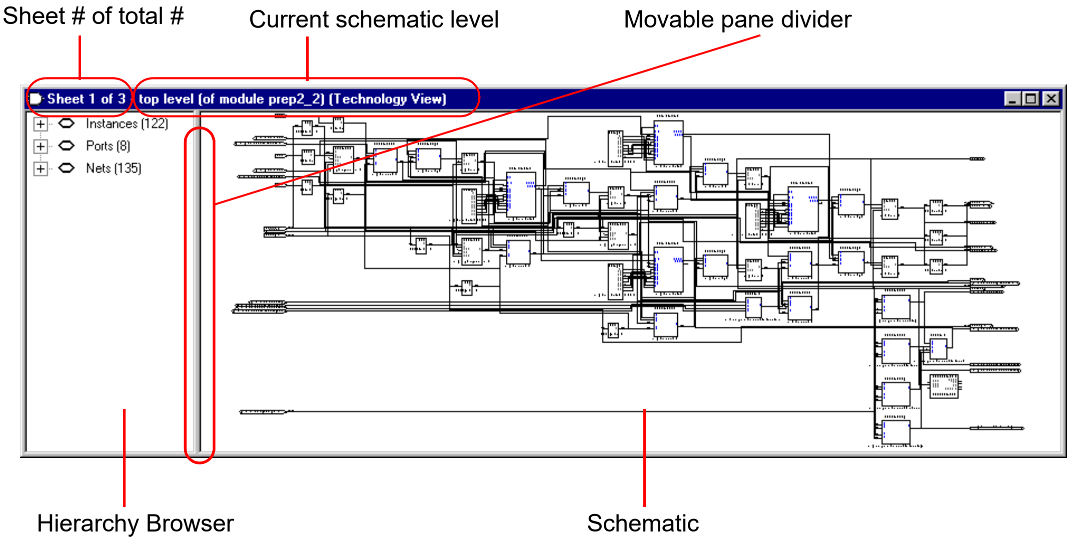
For information about ...	**See ...**
Hierarchy Browser	Hierarchy Browser, on page 52
Procedures for RTL view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Schematic, on page 248 of the *User Guide.*
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool, on page 87
Commands for RTL view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands, on page 89 HDL Analyst Menu, on page 452
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands, on page 341
History commands: Back and Forward	View Menu: RTL and Technology Views Commands, on page 341
Search command	Find Command (HDL Analyst), on page 332

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 49.

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 52. 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.



Sheet 1 of 3 top level (of module prep2_2) (Technology View)

Instances (122)
Ports (8)
Nets (135)

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 52
Procedures for Technology view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Schematic, on page 248 of the *User Guide*
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool, on page 87
Commands for Technology view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands, on page 89 HDL Analyst Menu, on page 452
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands, on page 341
History commands: Back and Forward	View Menu: RTL and Technology Views Commands, on page 341
Search command	Find Command (HDL Analyst), on page 332

Hierarchy Browser

The Hierarchy Browser is the left pane in the RTL and Technology views. (See *RTL View*, on page 49 and *Technology View*, on page 50.) 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 53 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 111* for more information about using the Hierarchy Browser for navigation and crossprobing.

Hierarchy Browser Symbols

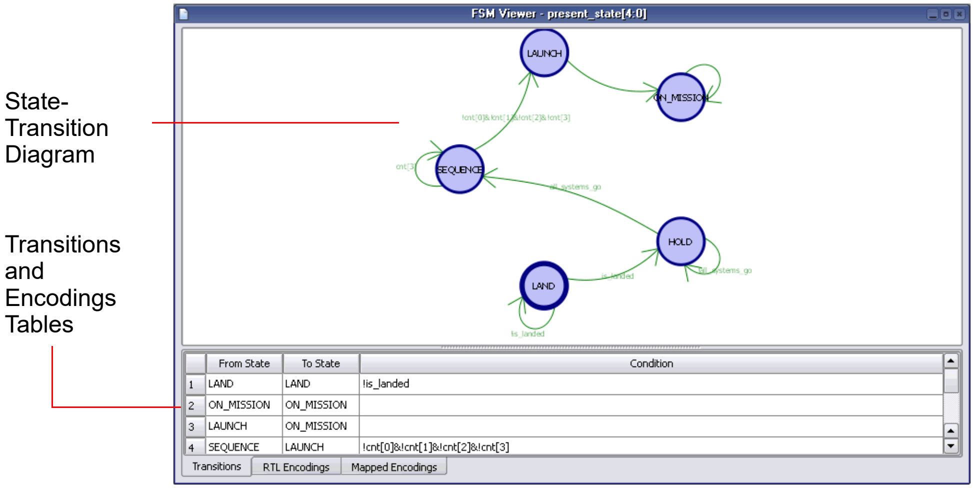
Common symbols used in Hierarchy Browsers are listed in the following table.

Symbol	Description	Symbol	Description
⬭	Folder	▷	Buffer
■	Input port	D	AND gate
🟨	Output port	D○	NAND gate
🟩	Bidirectional port	⊃	OR gate
⚬—⚬	Net	⊃○	NOR gate
□	Other primitive instance	⫐	XOR gate
▣	Hierarchical instance	⫐○	XNOR gate

Symbol	Description	Symbol	Description
⊏⊐	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.



State-Transition Diagram

Transitions and Encodings Tables

	From State	To State	Condition
1	LAND	LAND	!is_landed
2	ON_MISSION	ON_MISSION	
3	LAUNCH	ON_MISSION	
4	SEQUENCE	LAUNCH	!cnt[0]&!cnt[1]&!cnt[2]&!cnt[3]

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.

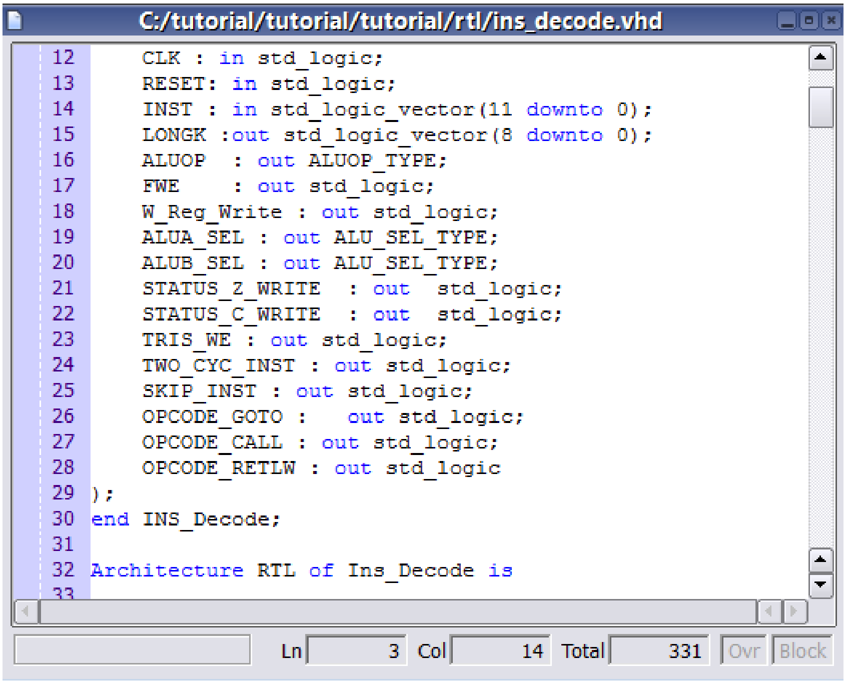
See also:

- *Pushing and Popping Hierarchical Levels, on page 108*, for information on the operation of pushing into a state machine.
- *FSM Viewer Toolbar, on page 74*, for information on the FSM icons.

- See *Using the FSM Viewer (Standard), on page 369* of the *User Guide* for more information on using the FSM viewer.

Text Editor View

The Text Editor view displays text files. These can be constraint files, source code files, or other informational or report files. You can enter and edit text in the window. You use this window to update source code and fix syntax or synthesis errors. You can also use it to crossprobe the design. For information about using the Text Editor, see *Editing HDL Source Files with the Built-in Text Editor, on page 48* in the *User Guide*.



```
    CLK : in std_logic;
    RESET: in std_logic;
    INST : in std_logic_vector(11 downto 0);
    LONGK :out std_logic_vector(8 downto 0);
    ALUOP  : out ALUOP_TYPE;
    FWE    : out std_logic;
    W_Reg_Write : out std_logic;
    ALUA_SEL : out ALU_SEL_TYPE;
    ALUB_SEL : out ALU_SEL_TYPE;
    STATUS_Z_WRITE  : out  std_logic;
    STATUS_C_WRITE  : out  std_logic;
    TRIS_WE : out std_logic;
    TWO_CYC_INST : out std_logic;
    SKIP_INST : out std_logic;
    OPCODE_GOTO :   out std_logic;
    OPCODE_CALL : out std_logic;
    OPCODE_RETLW : out std_logic
);
end INS_Decode;

Architecture RTL of Ins_Decode is
```

Ln 3 Col 14 Total 331

Opening the Text Editor

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

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

- 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.

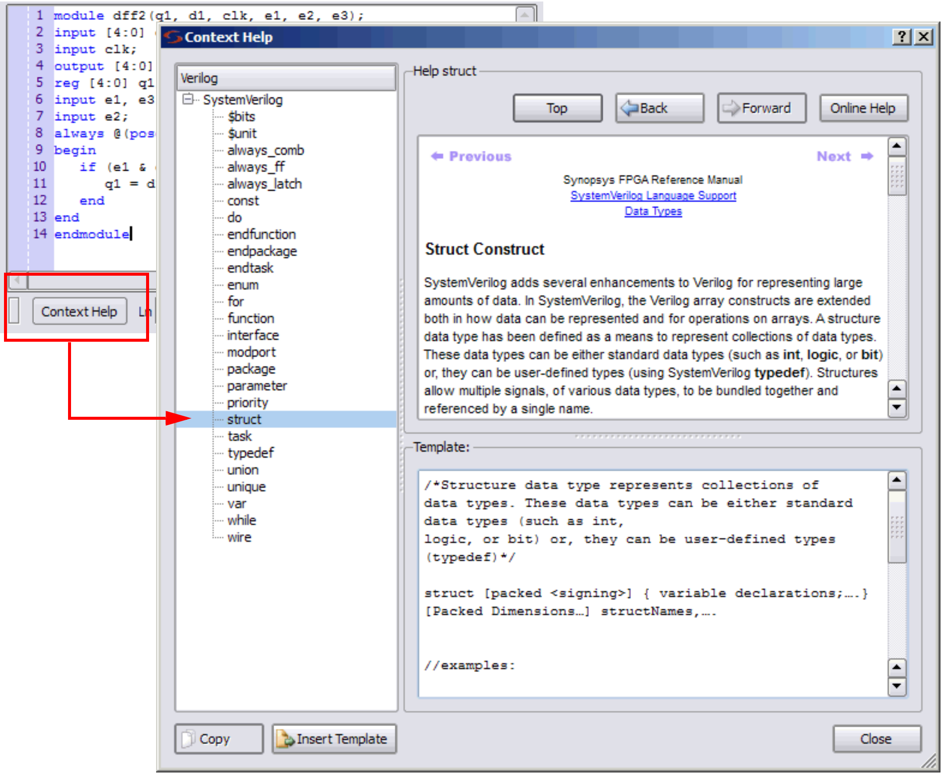
Feature	Description
Editing columns	Press and hold Alt, then drag the mouse down a column of text to select it.
Commenting out code	Choose Edit ->Advanced ->Comment Code. The rest of the current line is commented out: the appropriate comment prefix is inserted at the current text cursor position.
Checking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

See also:

- *Editor Options Command,* on page 470, for information on setting Text Editor preferences.
- *File Menu,* on page 322, for information on printing setup operations.
- *Text Editor Popup Menu,* on page 495, for information on the Text Editor popup menu.
- *Text Editor Toolbar,* on page 73, for information on bookmark icons of the Edit toolbar.
- *Keyboard Shortcuts,* on page 76, 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 on this button to display the Context Help Editor.



```
module dff2(q1, d1, clk, e1, e2, e3);
input [4:0]
input clk;
output [4:0]
reg [4:0] q1
input e1, e3
input e2;
always @(pos
begin
    if (e1 &
        q1 = d
    end
end
endmodule
```

Synopsys FPGA Reference Manual
SystemVerilog Language Support
Data Types

Struct Construct

SystemVerilog adds several enhancements to Verilog for representing large amounts of data. In SystemVerilog, the Verilog array constructs are extended both in how data can be represented and for operations on arrays. A structure data type has been defined as a means to represent collections of data types. These data types can be either standard data types (such as **int**, **logic**, or **bit**) or, they can be user-defined types (using SystemVerilog **typedef**). Structures allow multiple signals, of various data types, to be bundled together and referenced by a single name.

```
/*Structure data type represents collections of
data types. These data types can be either standard
data types (such as int,
logic, or bit) or, they can be user-defined types
(typedef)*/

struct [packed <signing>] { variable declarations;….}
[Packed Dimensions…] structNames,….


//examples:
```

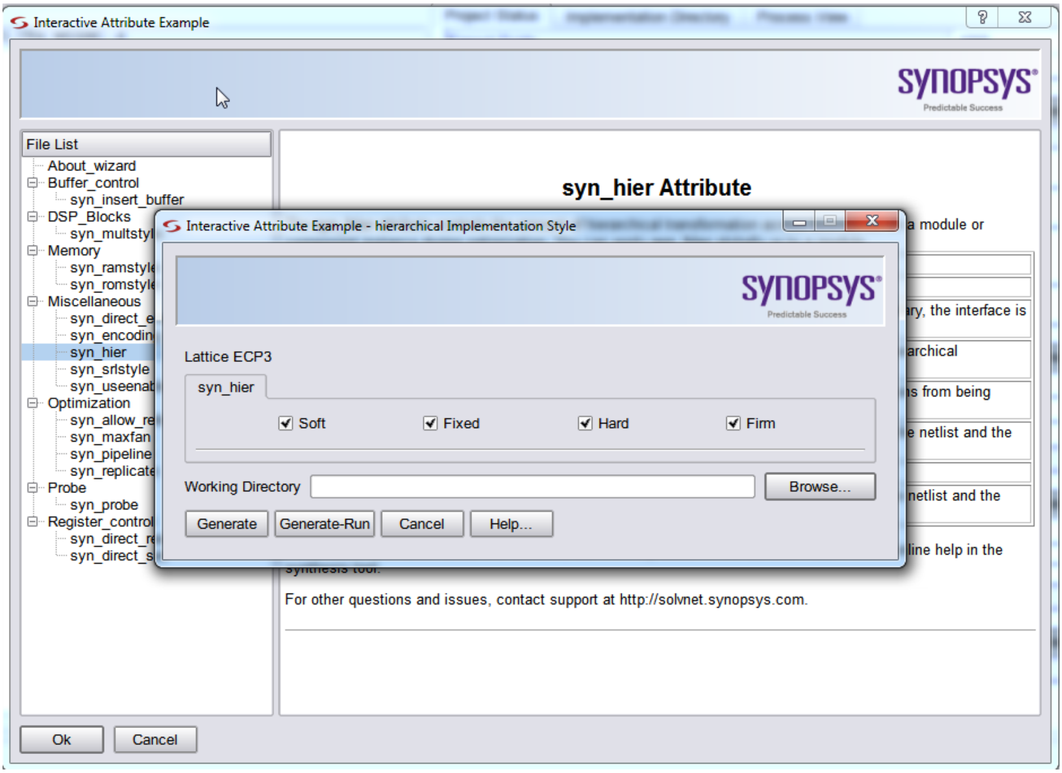
When you select a construct in the left-side of the window, the online help description for the construct is displayed. If the selected construct has this feature enabled, the online help topic is displayed on the top of the window and a generic code or command template for that construct is displayed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

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 Interactive Attribute Example in the Project view.



Lattice ECP3

☑ Soft ☑ Fixed ☑ Hard ☑ Firm

Working Directory

For other questions and issues, contact support at http://solnet.synopsys.com.

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

   A project will be created with an implementation for each attribute value selected.

5. 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.

FSM Compiler

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

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

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

During synthesis, a state encoding for an FSM is determined based on certain predefined characteristics of the state machine. You can force the use of a particular encoding style for a state machine by including the appropriate directive in the HDL description.

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

When to Use FSM Compiler

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

To view a textual description of a state machine in terms of inputs, states, and transitions, select the state machine in the RTL view, right-click, then choose View FSM Info File in the popup menu. You can view the same information graphically with the FSM viewer. The graphical description of a state machine makes it easier to verify behavior. For information on the FSM Viewer, see *FSM Viewer Window*, on page 54.

See also:

- *Log File*, on page 160, for information on the log file.
- *RTL and Technology Views Popup Menus*, on page 526, for information on the command View FSM Info File.

Where to Use FSM Compiler (Global and Local Use)

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

The following table summarizes the operations you can perform. For more information, see *Deciding when to Optimize State Machines*, on page 457 of the *User Guide.*

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

Using the Mouse

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

- Using Mouse Strokes, on page 65
- Using the Mouse Buttons, on page 66
- Using the Mouse Wheel, on page 68

Mouse Operation Terminology

The following terminology is used to refer to mouse operations:

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

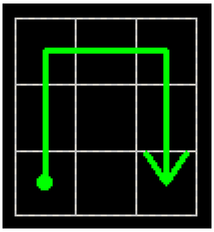
Using Mouse Strokes

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

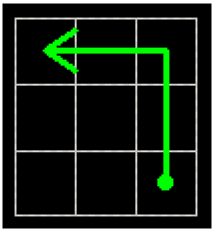
Some strokes are context sensitive. That is, the interpretation of the stroke depends upon the window in which the stroke is started. For example, in an Analyst view, the right stroke means “Next Sheet.” In a dialog box, the right stroke means “OK.”

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

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

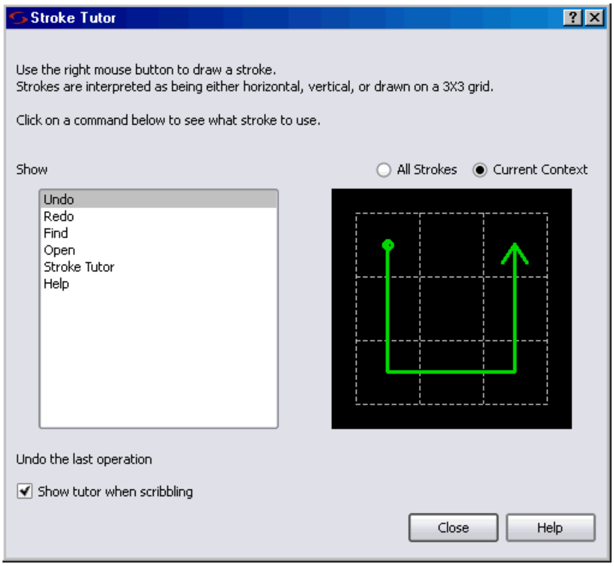


Back to Previous View

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)



Use the right mouse button to draw a stroke.
Strokes are interpreted as being either horizontal, vertical, or drawn on a 3X3 grid.

Click on a command below to see what stroke to use.

Show ○ All Strokes ● Current Context

- Undo
- Redo
- Find
- Open
- Stroke Tutor
- Help

Undo the last operation

☑ Show tutor when scribbling

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

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

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

Using the Mouse Buttons

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

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

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

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

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

  Right-clicking a selectable object in an HDL Analyst schematic also *selects* it, and deselects anything that was selected. The resulting popup menu applies only to the selected object. See *RTL View*, on page 49, and *Technology View*, on page 50, 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 RTL and Technology views.
- Text Editor Toolbar — Text Editor bookmark commands.
- FSM Viewer Toolbar — Display of finite state machine (FSM) information.
- Tools Toolbar — Opens supporting tools.

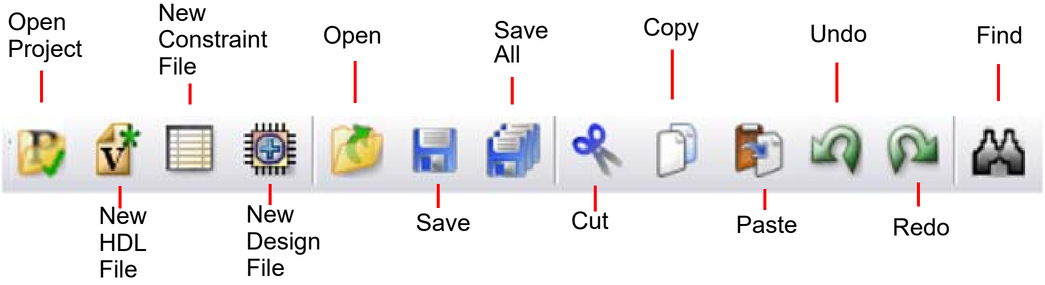
You can enable or disable the display of individual toolbars – see *Toolbar Command*, on page 343.

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:

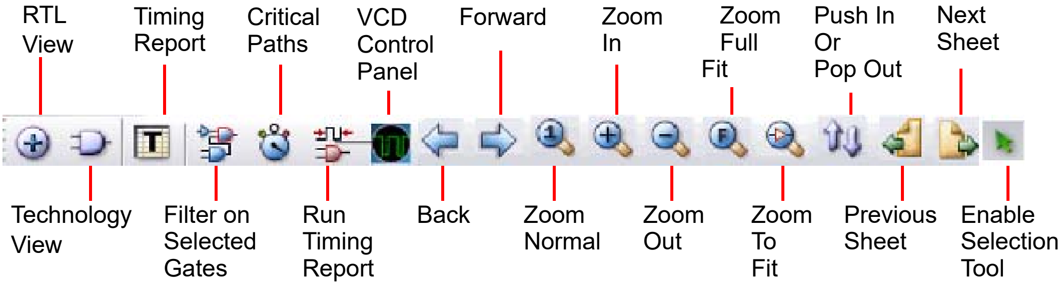
- *File Menu*, on page 322
- *Edit Menu*, on page 327

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).
Open	Displays the Open dialog box, to open a file. Same as File ->Open.
Save	Saves the current file. If the file has not yet been saved, this displays the Save As dialog box, where you specify the filename. The kind of file depends on the active view. Same as File ->Save.
Save All	Saves all files associated with the current design. Same as File ->Save All.
Cut	Cuts text or graphics from the active view, making it available to Paste. Same as Edit ->Cut.
Paste	Pastes previously cut or copied text or graphics to the active view. Same as Edit ->Paste.

Icon	Description
Undo	Undoes the last action taken. Same as Edit ->Undo.
Redo	Performs the action undone by Undo. Same as Edit ->Redo.
Find	Finds text in the Text Editor or objects in an RTL view or Technology view. Same as Edit ->Find.

Analyst Toolbar

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



The following table describes the default Analyst icons. Each is equivalent to an HDL Analyst menu command – see *HDL Analyst Menu*, on page 452, for more information.

Icon	Description
RTL View	Opens a new, hierarchical RTL view: a register transfer-level schematic of the compiled design, together with the associated Hierarchy Browser. Same as HDL Analyst ->RTL ->Hierarchical View.
Technology View	Opens a new, hierarchical Technology view: a technology-level schematic of the mapped (synthesized) design, together with the associated Hierarchy Browser. Same as HDL Analyst ->Technology ->Hierarchical View.
Timing Report View	Not applicable for Lattice technologies.
Filter Schematic	Filters your entire design to show only the selected objects. The result is a *filtered* schematic. Same as HDL Analyst ->Filter Schematic.
Show Critical Path	Filters your design to show only the instances (and their paths) whose slack times are within the slack margin of the worst slack time of the design (see HDL Analyst ->Set Slack Margin). The result is flat if the entire design was already flat. Icon Show Critical Path also enables HDL Analyst ->Show Timing Information. Available only in a Technology view. Not available in a Timing view. Same as HDL Analyst ->Show Critical Path.
Timing Analyst	Generates and displays a custom timing report and view. The timing report provides more information than the default report (specific paths or more than five paths) or one that provides timing based on additional analysis constraint files. See *Analysis Menu* , on page 440. Only available for certain device technologies. Same as Analysis ->Timing Analyst.
VCD Panel	Not applicable for Lattice technologies.
Back	Goes backward in the history of displayed sheets of the current HDL Analyst view. Same as View ->Back.

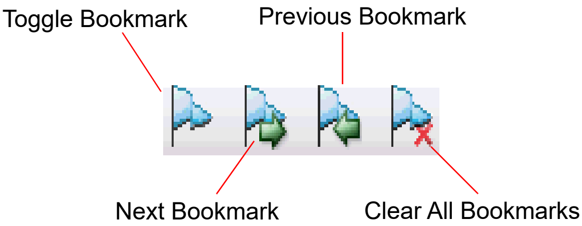
Icon	Description
Forward	Goes forward in the history of displayed sheets of the current HDL Analyst view. Same as View ->Forward.
Zoom 100%	Zooms in at a 1:1 ratio and centers the active view where you click. If the view is already normal size, it re-centers the view at the new click location. Same as View ->Normal View.[a]
Zoom In Zoom Out	Zooms the view in or out. Buttons stay active until deselected. Same as View ->Zoom In or View ->Zoom Out.[a]
Zoom Full	Zoom that reduces the active view to display the entire design. Same as View ->Full View.[b]
Zoom Selected	When selected, zooms in on only the selected objects to the full window size.
Push/Pop Hierarchy	Toggles traversing the hierarchy using the push/pop mode. Same as View ->Push/Pop Hierarchy.
Previous Sheet	Displays the previous sheet of a multiple-sheet schematic. Same as View ->Previous Sheet.
Next Sheet	Displays the next sheet of a multiple-sheet schematic. Same as View ->Previous Sheet.
Select Tool	Switches from zoom to the selection tool.

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

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

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 69.) The Edit toolbar provides the following icons, by default:

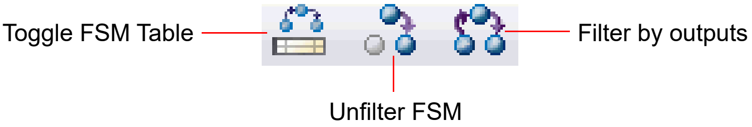


The following table describes the default Edit icons. Each is available in the Text Editor, and each is equivalent to an Edit menu command there.

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

FSM Viewer Toolbar

When you push down into a state machine primitive in an RTL view, the FSM Viewer displays and enables the FSM toolbar. 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 340, for more information.

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

Tools Toolbar

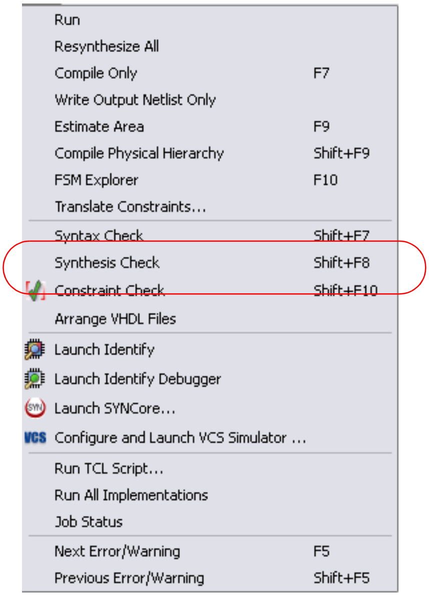
The Tools Toolbar opens supporting tools.

Icon	Description
Constraint Check	Checks the syntax and applicability of the timing constraints in the constraint file for your project and generates a report (*project_name*_cck.rpt). Same as Run->Constraint Check.
Launch 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 401.
VCS Simulator	Configures and launches the VCS simulator.

Keyboard Shortcuts

Keyboard shortcuts are key sequences that you type in order to run a command. Menus list keyboard shortcuts next to the corresponding commands.

For example, to 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 *filtered* schematic. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Expand Paths (see *HDL Analyst Menu: Filtering and Flattening Commands* , on page 455).
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 *filtered* schematic. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst ->Hierarchical ->Expand Paths (see *HDL Analyst Menu: Hierarchical and Current Level Submenus* , on page 453).
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 332.)
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 453).

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 *filtered* schematic. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical ->Expand (see *HDL Analyst Menu: Hierarchical and Current Level Submenus* , on page 453).
Ctrl-Enter (Return)	*I*n the FSM Viewer, hides all but the selected state(s). Same as View->Filter->Selected (see *View Menu* , on page 340).
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.
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. 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 341).
Ctrl-h	In the Text Editor, replaces text. Same as Edit->Replace (see *Edit Menu Commands for the Text Editor* , on page 328).
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 453).
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 453).

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 340).
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 453).
Ctrl-n	Creates a new file or project. Same as File->New.
Ctrl-o	Opens an existing file or project. Same as File->Open.
Ctrl-p	Prints the current view. Same as File->Print.
Ctrl-q	In an RTL or Technology view, toggles the display of visual properties of instances, pins, nets, and ports in a design.
Ctrl-r	In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, until registers, ports, or black boxes are reached. The result is a *filtered* schematic. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical->Expand to Register/Port (see *HDL Analyst Menu: Hierarchical and Current Level Submenus* , on page 453).
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 340).
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 328). 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 342).
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 341). 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 341). 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 328).
Ctrl-Shift-h	In an RTL or Technology view, shows all pins on selected *transparent* hierarchical (non-primitive) instances. Pins on primitives are always shown. Available only in a filtered schematic. Same as HDL Analyst ->Show All Hier Pins (see *HDL Analyst Menu: Analysis Commands* , on page 459).
Ctrl-Shift-i	In an RTL or Technology view, selects all instances on the current schematic level (all sheets). This does *not* select instances on other levels. Same as HDL Analyst->Select All Schematic->Instances (see *HDL Analyst Menu* , on page 452).
Ctrl-Shift-p	In an RTL or Technology view, selects all ports on the current schematic level (all sheets). This does *not* select ports on other levels. Same as HDL Analyst->Select All Schematic->Ports (see *HDL Analyst Menu* , on page 452).
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 328).

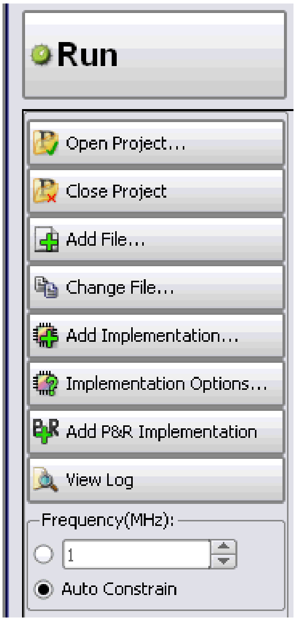
Keyboard Shortcut	Description
d	In an RTL or Technology view, selects the driver for the selected net. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Select Net Driver (see *HDL Analyst Menu , on page 452*).
Delete (DEL)	Removes the selected files from the project. Same as Project->Remove 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 452*).
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 341*). 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 328*).
F4	In the Project view, adds a file to the project. Same as Project->Add Source File (see *Build Project Command , on page 326*). 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 341*).
F5	Displays the next source file error. Same as Run->Next Error/Warning (see *Run Menu , on page 395*).
F7	Compiles your design, without mapping it. Same as Run->Compile Only (see *Run Menu , on page 395*).
F8	Synthesizes (compiles and maps) your design. Same as Run->Synthesize (see *Run Menu , on page 395*).
F10	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 341*).

Keyboard Shortcut	Description
F11	Toggles zooming in. Same as View->Zoom In (see *View Menu: RTL and Technology Views Commands* , on page 341).
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 455.
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 452).
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 452).
r	In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, until registers, ports, or black boxes are reached. The result is a *filtered* schematic. Limited to the current schematic. Same as HDL Analyst ->Current Level->Expand to Register/Port (see *HDL Analyst Menu* , on page 452).
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 395).
Shift-F7	Checks source file syntax. Same as Run->Syntax Check (see *Run Menu* , on page 395).
Shift-F8	Checks synthesis. Same as Run->Synthesis Check (see *Run Menu* , on page 395).
Shift-F10	Checks the timing constraints in the constraint files in your project and generates a report (*project_name*_cck.rpt). Same as Run->Constraint Check (see *Run Menu* , on page 395).

Keyboard Shortcut	Description
Shift-F11	Toggles zooming out. Same as View->Zoom Out (see *View Menu* , on page 340).
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.

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.



Frequency(MHz):
○ 1
● Auto Constrain

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 327).
Close Project	Closes the current project. Same as File->Close Project (see *Run Menu* , on page 395).
Add File...	Adds a source file to the project. Same as Project->Add Source File (see *Build Project Command* , on page 326).

Button/Option	Action
Change File...	Replaces one source file with another. Same as Project ->Change File (see *Change File Command* , on page 352).
Add Implementation	Creates a new implementation.
Implementation Options	Displays the Implementation Options dialog box, where you can set various options for synthesis.
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 514 for a description of the dialog box, and *Running P&R Automatically after Synthesis* , on page 652in the *User Guide* for information about using this feature.
View Log	Displays the log file. Same as View ->View Log File (see *View Menu* , on page 340).
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.
Auto Constrain	When Auto Constrain is enabled and no clocks are defined, the software automatically constrains the design to achieve best possible timing by reducing periods of individual clock and the timing of any timed I/O paths in successive steps. See *Using Auto Constraints* , on page 393 in the *User Guide* for detailed information about using this option. You can also set this option on the Constraints panel of the Implementation Options dialog box.
FSM Compiler	Turning on this option enables special FSM optimizations. Same as enabling the FSM Compiler option on the Options panel of the Implementation Options dialog box (see *FSM Compiler* , on page 62 and *Optimizing State Machines* , on page 457 in the *User Guide*).

Button/Option	Action
Resource Sharing	When enabled, makes the compiler use resource sharing techniques. This option does not affect resource sharing by the mapper. The option is the same as the Resource Sharing option on the Options panel of the Implementation Options dialog box. See *Sharing Resources* , on page 454 in the *User Guide* for usage details.
Pipelining	When enabled, uses register balancing and pipeline registers on multipliers and ROMs. Same as enabling the Pipelining option on the Options panel of the Implementation Options dialog box. See *Pipelining* , on page 434in the *User Guide*.
Retiming	When enabled, improves the timing performance of sequential circuits. The retiming process moves storage devices (flip-flops) across computational elements with no memory (gates/LUTs) to improve the performance of the circuit. This option also adds a retiming report to the log file. Same as enabling the Retiming option on the Options panel of the Implementation Options dialog box. Use the syn_allow_retiming attribute to enable or disable retiming for individual flip-flops. See *syn_allow_retiming* , on page 36 for syntax details. Note: Pipelining is automatically enabled when retiming is enabled.
Run	Runs synthesis (compilation and mapping). Same as the Run->Synthesize command (see *Run Menu* , on page 395).

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 following describe the HDL Analyst tool and the operations you can perform with it.

- HDL Analyst Views and Commands, on page 88
- Schematic Objects and Their Display, on page 90
- Basic Operations on Schematic Objects, on page 99
- Multiple-sheet Schematics, on page 105
- Exploring Design Hierarchy, on page 108
- Filtering and Flattening Schematics, on page 115
- Timing Information and Critical Paths, on page 121

For additional information, see the following:

- Descriptions of the HDL Analyst commands in Chapter 2, *User Interface Overview*:
- Chapter 13, *Optimizing Processes for Productivity* in the *User Guide*

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 49 and *Technology View*, on page 50 for information). The graphic representation is useful for analyzing and debugging your design, because you can visualize where coding changes or timing constraints might reduce area or increase performance.

This section gives you information about the following:

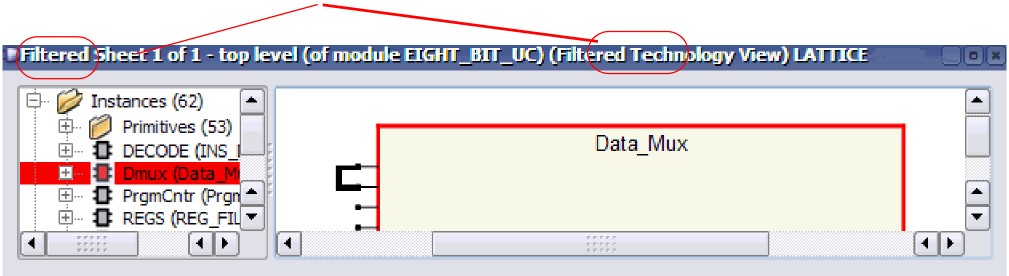
- Filtered and Unfiltered Schematic Views, on page 88
- Accessing HDL Analyst Commands, on page 89

Filtered and Unfiltered Schematic Views

HDL Analyst views (*RTL View*, on page 49 and *Technology View*, on page 50) consist of schematics that let you analyze your design graphically. The schematics can be filtered or unfiltered. The distinction is important because the kind of view determines how objects are displayed for certain commands.

- Unfiltered schematics display all the objects in your design, at appropriate hierarchical levels.
- Filtered schematics show only a subset of the objects in your design, because the other objects have been filtered out by some operation. The Hierarchy Browser in the filtered view always list all the objects in the design, not just the filtered objects. Some commands, such as HDL Analyst -> Show Context, are only available in filtered schematics. Views with a filtered schematic have the word Filtered in the title bar.

Indicates a filtered schematic


Filtered Sheet 1 of 1 - top level (of module EIGHT_BIT_UC) (Filtered Technology View) LATTICE
Instances (62)
Primitives (53)
DECODE (INS_I
Dmux (Data_M
PrgmCntr (Prgm
REGS (REG_FIL
Data_Mux

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 115.
- For procedures on using filtering, see *Filtering Schematics*, on page 356 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 452
- View Menu, on page 340
- RTL and Technology Views Popup Menus, on page 526
- Analyst Toolbar, on page 71

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 90
- Sheet Connectors, on page 91
- Primitive and Hierarchical Instances, on page 92
- Hidden Hierarchical Instances, on page 95
- Transparent and Opaque Display of Hierarchical Instances, on page 93
- Schematic Display, on page 95

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

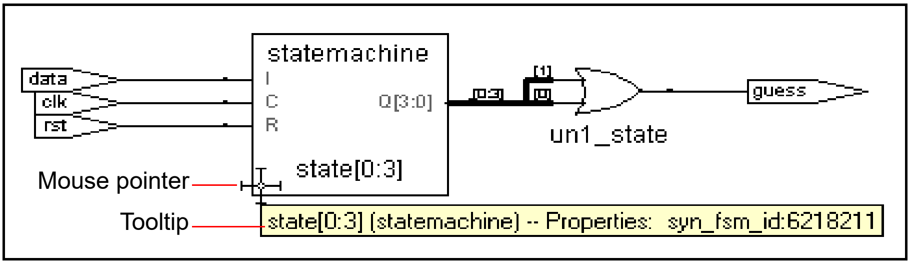
To obtain information about specific objects, you can view object properties with the Properties command from the right-click popup menu, or place the pointer over the object and view the object information displayed. With the latter method, information about the object displays in these two places until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



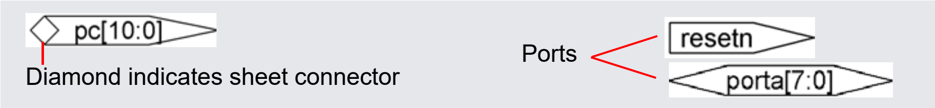
  To disable tooltip display, select View -> Toolbars and disable the Show Tooltips option. Do this if you want to reduce clutter.

See also

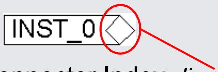
- Pin and Pin Name Display for Opaque Objects, on page 96
- HDL Analyst Options Command, on page 477

Sheet Connectors

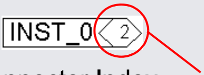
When the HDL Analyst tool divides a schematic into multiple sheets, sheet connector symbols indicate how sheets are related. A sheet connector symbol is like a port symbol, but it has an empty diamond with sheet numbers at one end. Use the Options->HDL Analyst Options command (see *Sheet Size Panel*, on page 484) 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.



Show Sheet Connector Index *disabled*



Show Sheet Connector Index *enabled*

See also

- Multiple-sheet Schematics, on page 105
- HDL Analyst Options Command, on page 477
- RTL and Technology Views Popup Menus, on page 526

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 Symbols,* on page 53). 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.

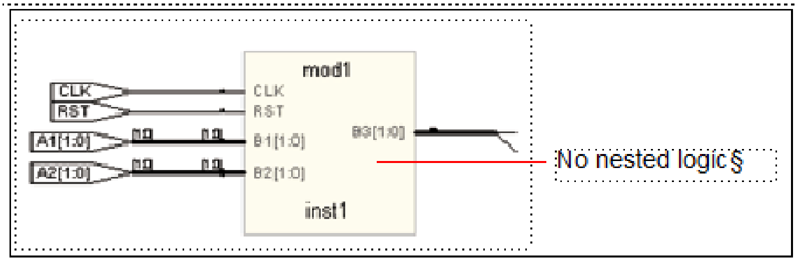
The Hierarchy Browser lists hierarchical instances under Instances, and uses this symbol: ▣ . In a schematic, the display of hierarchical instances depends on the combination of the following:

- 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 93 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 95 for more information.

Transparent and Opaque Display of Hierarchical Instances

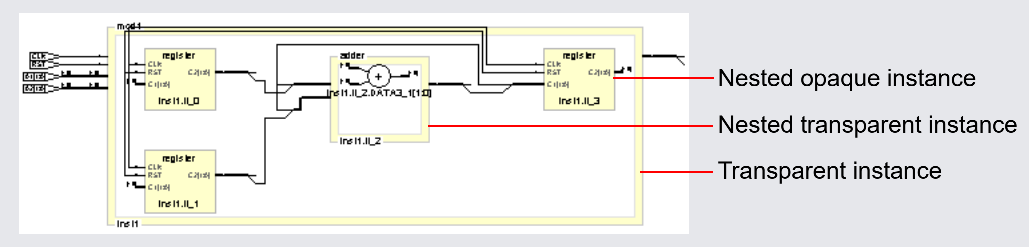
A hierarchical instance can be displayed transparently or opaquely. You cannot directly control the display; certain commands cause instances to be transparent. The distinction between transparent and opaque is important because some commands operate differently on transparent and opaque instances. For 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 113* 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.



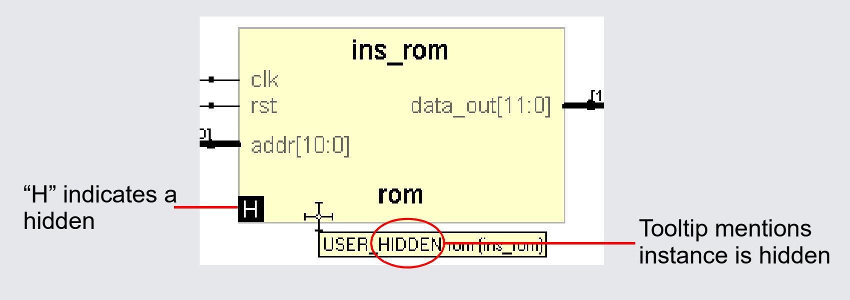
See also

- Looking Inside Hierarchical Instances, on page 113
- Multiple Sheets for Transparent Instance Details, on page 107
- Filtered and Unfiltered Schematic Views, on page 88

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 526). 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.



“H” indicates a hidden

Tooltip mentions instance is hidden

Schematic Display

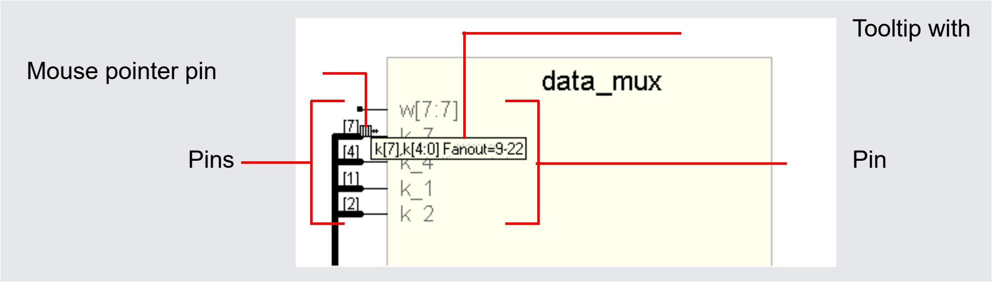
The HDL Analyst Options dialog box controls general properties for all HDL Analyst views, and can determine the display of schematic object information. Setting a display option affects all objects of the given type in all views. Some schematic options only take effect in schematic windows opened after the setting change; others affect existing schematic windows as well.

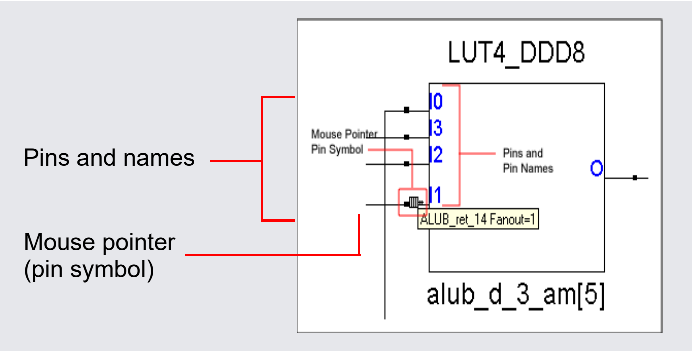
The following are some commonly used settings that affect the display of schematic objects. See *HDL Analyst Options Command*, on page 477 for a complete list of display options.

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 Filtered Instances Instances added for expansion	Instances on a schematic by setting limits to the number of instances displayed
Instance Name Show Conn Name Show Symbol Name Show Port Name	Object labels
Show Pin Name HDL Analyst->Show All Hier Pins	Pin names. See *Pin and Pin Name Display for Opaque Objects* , on page 96 and *Pin and Pin Name Display for Transparent Objects* , on page 97 for details.

Pin and Pin Name Display for Opaque Objects

Although it always displays the pins, the software does not automatically display pin names for opaque hierarchical instances, technology-specific primitives, RAMS, ROMs, and state machines. To display pin names for these objects, enable Options-> HDL Analyst Options->Text->Show Pin Name. The following figures illustrate this display. The first figure shows pins and pin names of an opaque hierarchical instance, and the second figure shows the pins of a technology-specific primitive with its cell contents not displayed.





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.

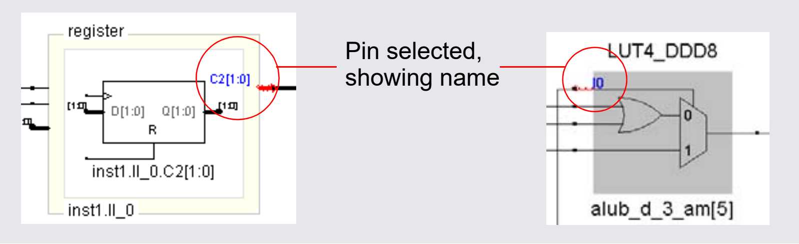
Transparent Hierarchical Instances

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.



Pin selected, showing name

See also:

- HDL Analyst Options Command, on page 477
- Controlling the Amount of Logic on a Sheet, on page 105
- Analyzing Timing in Schematic Views, on page 376 in the *User Guide*

Basic Operations on Schematic Objects

Basic operations on schematic objects include the following:

- Finding Schematic Objects, on page 99
- Selecting and Unselecting Schematic Objects, on page 100
- Crossprobing Objects, on page 102
- Dragging and Dropping Objects, on page 103

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

- Filtering and Flattening Schematics, on page 115
- Timing Information and Critical Paths, on page 121
- Multiple-sheet Schematics, on page 105
- Exploring Design Hierarchy, on page 108

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 331 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 331 in the *User Guide* for details.
- Edit -> Find command

  The Edit -> Find command is described in *Find Command (HDL Analyst)*, on page 332. 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 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 333* 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 115* 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 316* 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 crosshairs 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 99
- HDL Analyst Menu, on page 452

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 344.

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	The RTL view or Technology view must be open. Select the code in the Text Editor that corresponds to the object(s) you want to crossprobe. The object corresponding to the selected code is automatically selected in the target view, if an HDL source file is in the Text Editor. Otherwise, right-click and choose the Select in Analyst command. To cross-probe from text other than source code, first select Options->HDL Analyst Options and then enable Enhanced Text Crossprobing.

From ...	To ...	Do this ...
FSM Viewer	Analyst view	The target view must be open. The state machine must be encoded with the onehot style to crossprobe from the transition table. Select a state anywhere in the FSM Viewer (bubble diagram or transition table). The corresponding object is automatically selected in the HDL Analyst view.
Analyst view FSM Viewer	Text Editor	Double-click an object. The source code corresponding to the object is automatically selected in the Text Editor, which is opened to show the selection. If you just select an object, without double-clicking it, the corresponding source code is still selected and displayed in the editor (provided it is open), but the editor window is not raised to the front.
Analyst view	Another open view	Select an object in an HDL Analyst view. The object is automatically selected in all open views. If the target view is the FSM Viewer, 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 91) mark logic connections from one sheet to the next.

For more information, see

- Controlling the Amount of Logic on a Sheet, on page 105
- Navigating Among Schematic Sheets, on page 105
- Multiple Sheets for Transparent Instance Details, on page 107

Controlling the Amount of Logic on a Sheet

You can control the amount of logic on a schematic sheet using the options in Options->HDL Analyst Options->Sheet Size. The Maximum Instances option sets the maximum number of instances on an unfiltered schematic sheet. The Maximum Filtered Instances option sets the maximum number of instances displayed at any given hierarchical level on a filtered schematic sheet.

See also:

- HDL Analyst Options Command, on page 477
- Setting Schematic Preferences, on page 320 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.

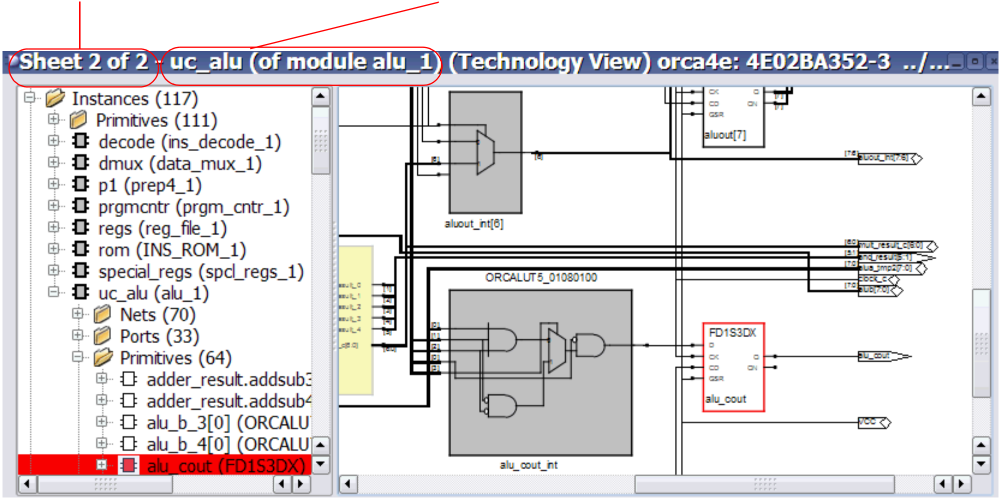
Multisheet Orientation in the Title Bar

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.

Sheet # of total #

Context (level) of current sheet: instance name and module



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 317* 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 115
- View Menu: RTL and Technology Views Commands, on page 341
- Pushing and Popping Hierarchical Levels, on page 108

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 105.
- 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 105
- View Menu: RTL and Technology Views Commands, on page 341

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 108
- Navigating With a Hierarchy Browser, on page 111
- Looking Inside Hierarchical Instances, on page 113

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 111 and *Looking Inside Hierarchical Instances*, on page 113 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:

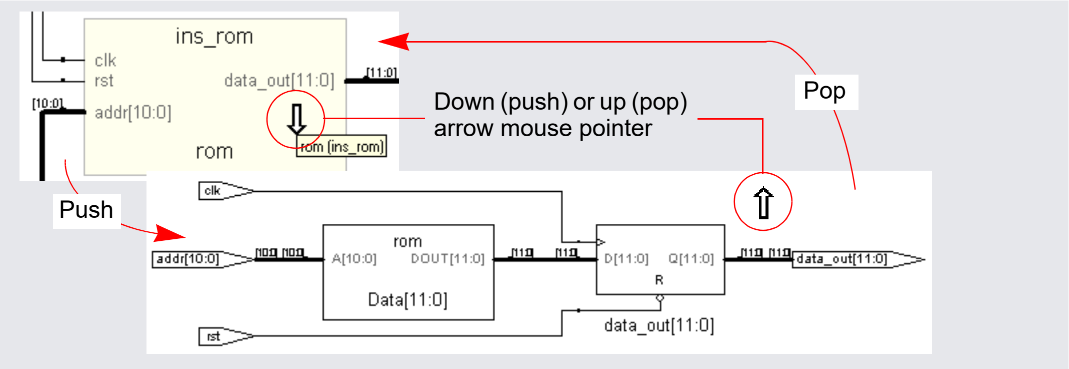
- When you push into an inferred state machine in an RTL view, the FSM Viewer opens, with graphical information about the FSM. See the *FSM Viewer Window,* on page 54, for more information.
- When you push into an inferred ROM in an RTL view, the Text Editor window opens and displays the ROM data table (`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 109 for details.
- A small H symbol (H) in the lower left corner indicates a hidden instance, and you cannot push into it.
- The Hierarchy Browser symbols indicates the type of instance and you can use that to determine whether you can push into an object. For example, hierarchical instance (▣), technology-specific primitive (⊏), logic primitive such as XOR (⊐), or other primitive instance (□). The browser symbol does not indicate whether or not an instance is hidden.
- The *status bar* at the bottom of the main synthesis tool window reports information about the object under the pointer, including whether or not it is a hidden instance or a primitive.

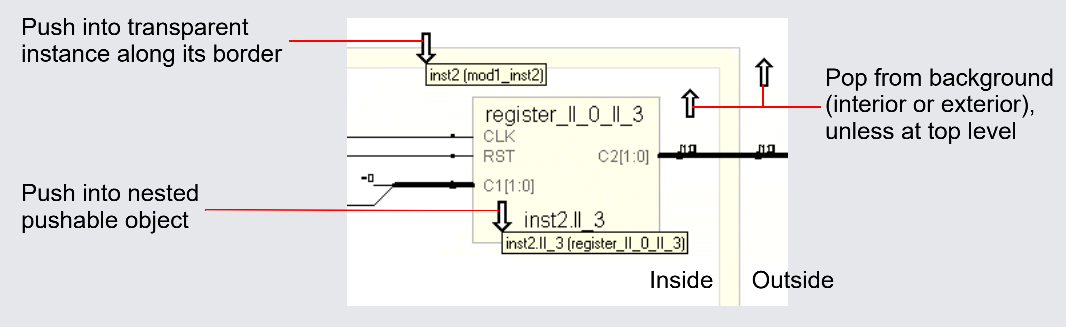
How to Push and Pop Hierarchical Levels

You push/pop design levels with the HDL Analyst Push/Pop mode. To enable or disable this mode, toggle View->Push/Pop Hierarchy, use the icon, or use the appropriate mouse strokes.



Once Push/Pop mode is enabled, you push or pop as follows:

- To *pop*, place the pointer in an empty area of the schematic background, then click or use the appropriate mouse stroke. The background area inside a transparent instance acts just like the background area outside the instance.
- To *push* into an object, place the mouse pointer over the object and click or use the appropriate mouse stroke. To push into a transparent instance, place the pointer over its pale yellow border, not its hollow (white) interior. Pushing into an object nested inside a transparent hierarchical instance descends to a lower level than pushing into the enclosing transparent instance. In the following figure, pushing into transparent instance inst2 descends one level; pushing into nested instance inst2.II_3 descends two levels.


Push into transparent instance along its border
inst2 (mod1_inst2)
register_II_0_II_3
CLK
RST
C2[1:0]
C1[1:0]
inst2.II_3
inst2.II_3 (register_II_0_II_3)
Pop from background (interior or exterior), unless at top level
Push into nested pushable object
Inside
Outside

The following arrow mouse pointers indicate status in Push/Pop mode. For other indicators, see *Pushable Schematic Objects*, on page 108.

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:

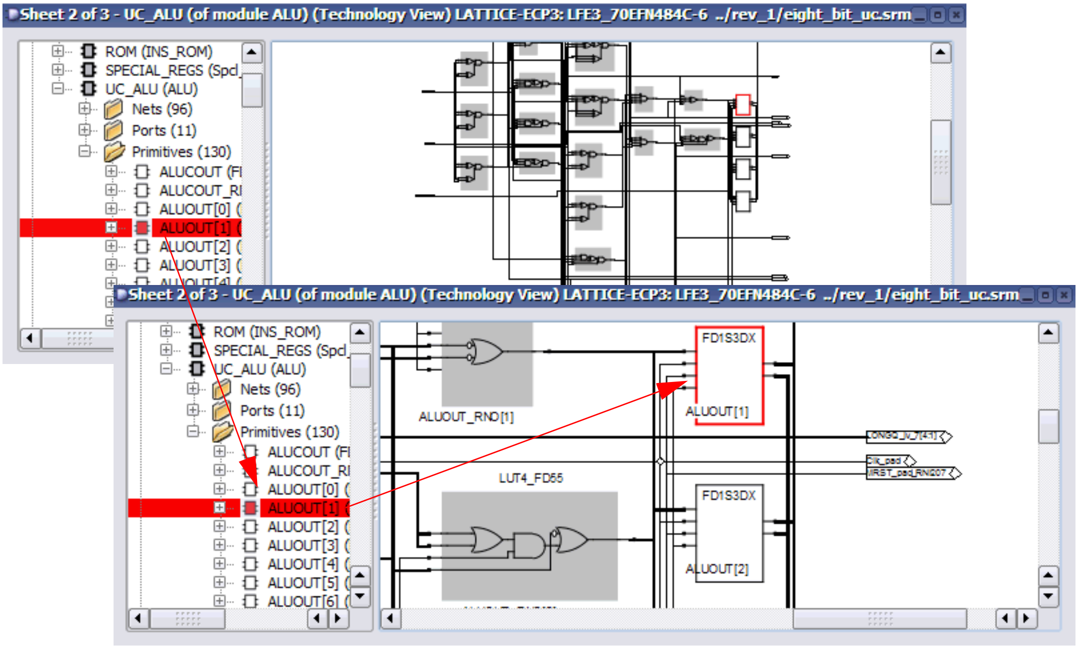
- Hidden Hierarchical Instances, on page 95
- Transparent and Opaque Display of Hierarchical Instances, on page 93
- Using Mouse Strokes, on page 65
- Navigating With a Hierarchy Browser, on page 111

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 108 and *Looking Inside Hierarchical Instances*, on page 113 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.



Selecting an object in the browser causes the schematic to display it also.

Explore the browser hierarchy by expanding or collapsing the categories in the browser. You can also use the arrow keys (left, right, up, down) to move up and down the hierarchy and select objects. To select more than one object, press Ctrl and select the objects in the browser. To select a range of schematic objects, click an object at one end of the range, then hold the Shift key while clicking the name of an object at the other end of the range.

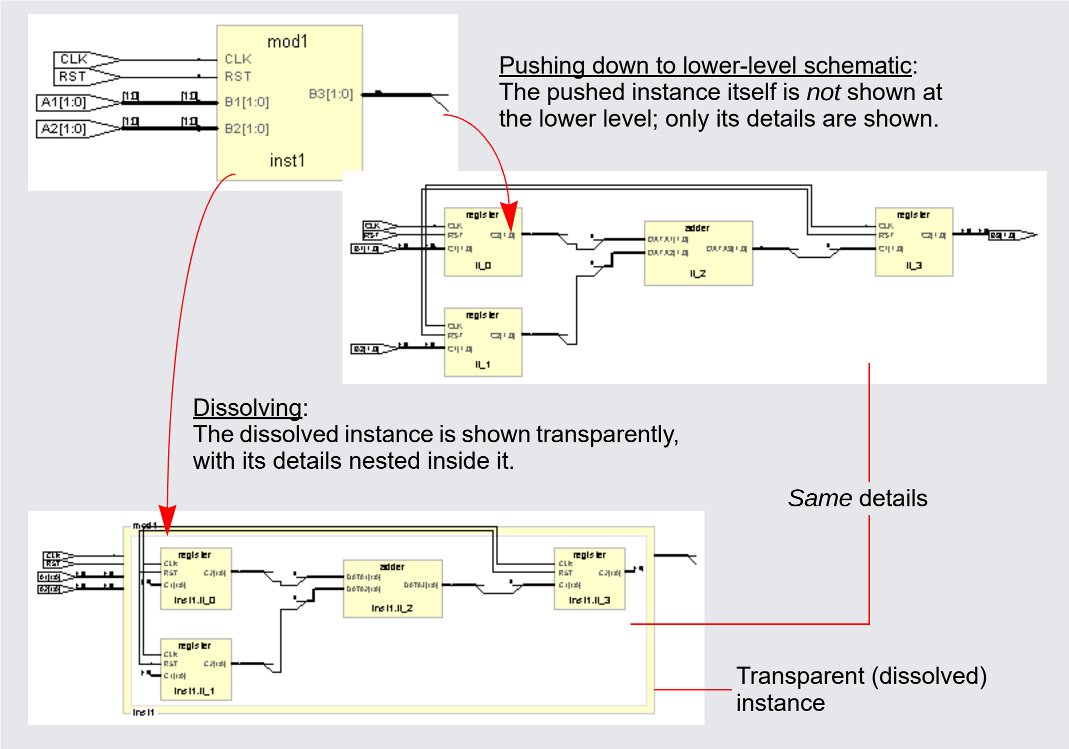
See also:

- Crossprobing Objects, on page 102
- Pushing and Popping Hierarchical Levels, on page 108
- Hierarchy Browser Popup Menu Commands, on page 526

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 111 and *Navigating With a Hierarchy Browser*, on page 111 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 lower-level logic a level down. The following figure compares the same lower-level logic viewed in a transparent instance and a push operation:



Pushing down to lower-level schematic: The pushed instance itself is *not* shown at the lower level; only its details are shown.

Dissolving: The dissolved instance is shown transparently, with its details nested inside it.

Same details

Transparent (dissolved) instance

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 115 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 108
- Navigating With a Hierarchy Browser, on page 111
- HDL Analyst Options Command, on page 477
- Transparent and Opaque Display of Hierarchical Instances, on page 93
- Hidden Hierarchical Instances, on page 95

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 115
- Combined Filtering Operations, on page 116
- Returning to The Unfiltered Schematic, on page 117
- Commands That Flatten Schematics, on page 117
- Selective Flattening, on page 118
- Filtering Compared to Flattening, on page 120

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 356 in the *User Guide*.

See also:

- Filtered and Unfiltered Schematic Views, on page 88
- HDL Analyst Menu, on page 452 and RTL and Technology Views Popup Menus, on page 526

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 455 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 453 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 453 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 459 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 99 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 526
- View Menu: RTL and Technology Views Commands, on page 341

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.	

Command	Unfiltered Schematic	Filtered Schematic
Technology-> Flattened View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of technology cells.	
Technology-> Flattened to Gates View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of Boolean logic gates.	
Technology-> Flattened Critical Path	Creates a filtered, flattened Technology view schematic that shows only the instances with the worst slack times and their path.	
Unflatten Schematic	Undoes any flattening done by Dissolve Instances and Flatten Current Schematic at the current schematic level. Returns to the original schematic, as it was before flattening (and any filtering).	

All the commands are on the HDL Analyst menu except Unflatten Schematic, which is available in a schematic popup menu.

The most versatile commands, are Dissolve Instances and Flatten Current Schematic, which you can also use for selective flattening (*Selective Flattening*, on page 118).

See also:

- Filtering Compared to Flattening, on page 120
- Selective Flattening, on page 118

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 526 for descriptions):

- Use Hide Instances to hide instances that you do *not* want to flatten, then flatten the others (flattening operations do not recognize hidden instances). After flattening, you can Unhide Instances that are hidden.
- Flatten selected hierarchical instances using one of these commands:
  - If the current schematic is unfiltered, use Dissolve Instances.

– If the schematic is filtered, use Dissolve Instances, followed by Flatten Current Schematic. In a filtered schematic, Dissolve Instances makes the selected instances transparent and Flatten Current Schematic flattens only transparent instances.

The Dissolve Instances and Flatten Current Schematic (or Flatten Schematic) commands behave differently in filtered and unfiltered schematics as outlined in the following table:

Command	**Unfiltered Schematic**	**Filtered Schematic**
Dissolve Instances	Flattens selected instances	Provides virtual flattening: makes selected instances transparent, displaying their lower-level details.
Flatten Current Schematic Flatten Schematic	Flattens *everything* at the current level and below	Flattens only the non-hidden, *transparent* hierarchical instances: does not flatten opaque or hidden instances. See below for details of the process.

In a filtered schematic, flattening with Flatten Current Schematic is actually a two-step process:

1. The transparent instances of the schematic are flattened in the context of the entire design. The result of this step is the entire hierarchical design, with the transparent instances of the filtered schematic replaced by their internal logic.
2. The original filtering is then restored: the design is refiltered to show only the logic that was displayed before flattening.

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 526
- Selective Flattening, on page 118

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 121
- Critical Paths and the Slack Margin Parameter, on page 122
- Examining Critical Path Schematics, on page 124

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

- Watch Window, on page 40
- Log File, on page 160
- Chapter 13, *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 370). 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 440
- Timing Reports, on page 165
- Log File, on page 160

Critical Paths and the Slack Margin Parameter

The HDL Analyst tool can isolate critical paths in your design, so that you can analyze problem areas, add timing constraints where appropriate, and resynthesize for better results.

After you successfully run synthesis, you can display just the critical paths of your design using any of the following commands from the HDL Analyst menu:

- Hierarchical Critical Path
- Flattened Critical Path
- Show Critical Path

The first two commands create a new Technology view, hierarchical or flattened, respectively. The Show Critical Path command reuses the current Technology view. Neither the current selection nor the current sheet display have any effect on the result. The result is flat if the entire design was already flat; otherwise it is hierarchical. Use Show Critical Path if you want to maintain the existing display history.

All these commands filter your design to show only the instances (and their paths) with the worst slack times. They also enable HDL Analyst -> Show Timing Information, displaying timing information.

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 452
- HDL Analyst Options Command, on page 477
- Handling Negative Slack, on page 382 of the *User Guide*
- Analyzing Timing in Schematic Views, on page 376 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 117
- Filtering and Flattening Schematics, on page 115

CHAPTER 4

Constraints

Constraints are used in the FPGA synthesis environment to achieve optimal design results. Timing constraints set performance goals, non-timing constraints (design constraints) guide the tool through optimizations that further enhance performance

This chapter provides an overview of how constraints are handled in the FPGA synthesis environment.

- Constraint Types, on page 126
- Constraint Files, on page 127
- Timing Constraints, on page 129
- FDC Constraints, on page 132
- Methods for Creating Constraints, on page 133
- Constraint Translation, on page 135
- Constraint Checking, on page 140
- Database Object Search, on page 142
- Forward Annotation, on page 143
- Auto Constraints, on page 143

Constraint Types

One way to ensure the FPGA synthesis tools achieve the best quality of results for your design is to define proper constraints. In the FPGA environment, constraints can be categorized by the following types:

Type	Description
Timing	Performance constraints that guide the synthesis tools to achieve optimal results. Examples: clocks (create_clock), clock groups (set_clock_groups), and timing exceptions like multicycle and false paths (set_multicycle_path...) See *Timing Constraints* , on page 129 for information on defining these constraints.
Design	Additional design goals that enhance or guide tool optimizations. Examples: Attributes and directives (define_attribute, define_global_attribute), I/O standards (define_io_standard), and compile points (define_compile_point).
Physical	Constraints that provide placement optimizations for synthesis.

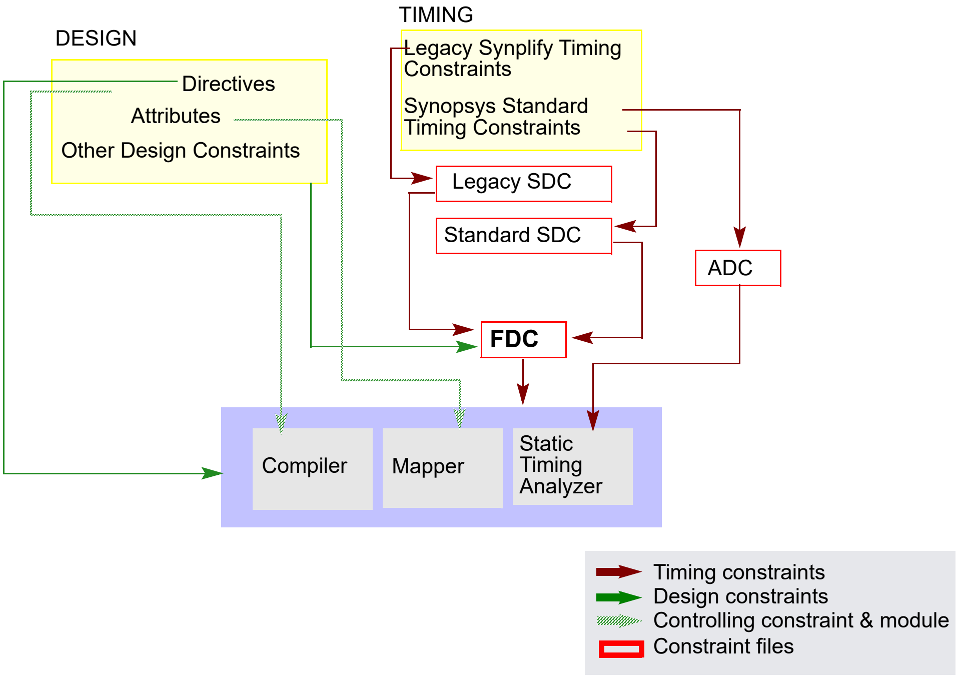
The easiest way to specify constraints is through the SCOPE interface. The tool saves timing and design constraints to an FDC file that you add to your project.

See Also

Constraint Files , on page 127	Overview of constraint files
Timing Constraints , on page 129	Overview of timing constraint definitions and FDC file generation.
SCOPE Constraints Editor , on page 230	Information about automatic generation of timing and design constraints.
Timing Constraints, on page 278	Timing constraint syntax
Design Constraints , on page 317	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 129. 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.
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 tools have supported different timing formats in the past, and this section describes some of the details of standardization:

- Legacy SDC and Synopsys Standard SDC, on page 129
- FDC File Generation, on page 130
- Timing Constraint Precedence in Mixed Constraint Designs, on page 131

Legacy SDC and Synopsys Standard SDC

Releases prior to G-2012.09 had two types of constraint files that could be used in a design project:

- Legacy “Synplify-style” timing constraints (define_clock, define_false_path...) saved to an sdc file. This file also included non-timing design constraints, like attributes and compile points.
- Synopsys standard timing constraints (create_clock, set_false_path…). These constraints were also saved to an sdc file, which only contained timing constraints. Non-timing constraints were in a separate sdc file. The tool used the two files together, drawing timing constraints from one and non-timing constraints from the other.

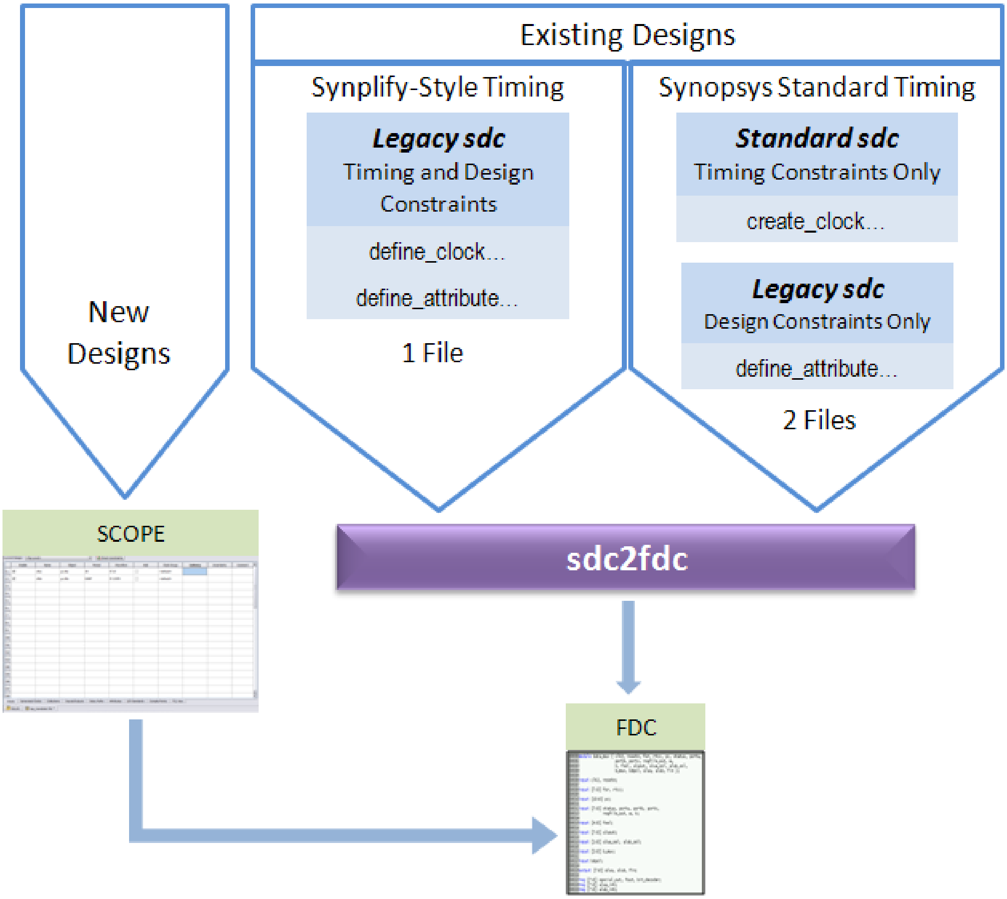
Starting with the G-2012.09 release, Synopsys standard timing constraint format has replaced the legacy-style constraint format, and a new FDC (FPGA design constraint) file consolidates both timing and design formats. As a result of these updates, there are some changes in the use model:

- Timing constraints in the legacy format are converted and included in an FDC file, which includes both timing and non-timing constraints. The file uses the Synopsys standard syntax for timing constraints (create_-clock, set_multicyle_path...). The syntax for non-timing design constraints is unchanged (define_attribute, define_io_standard…).
- The SCOPE editor has been enhanced to support the timing constraint changes, so that new constraints can be entered correctly.
- For older designs, use the sdc2fdc command to do a one-time conversion.

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 278.

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 133 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 191 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	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 5, *SCOPE Constraints Editor*, for information how to use SCOPE to automatically generate constraint syntax. Access: File->New->FPGA Design Constraints ...
Manually-Entered Text Editor (fdc File, all other constraint files)	You can manually enter constraints in a text file. Make sure to use the correct syntax for the timing and design commands. The SCOPE GUI includes a TCL View with an advanced text editor, where you can manually generate the constraint syntax. For a description of this view, see TCL View, on page 179. You can also open any constraint file in a text editor to modify it.
Source Code Attributes/Directives (HDL files)	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. Attributes can be entered through the SCOPE interface, as they affect the mapper, not the compiler
Automatic— First Pass	Enable the Auto Constrain button in the Project view to have the tool automatically generate constraints based on inferred clocks. See *Using Auto Constraints* , on page 393 in the *User Guide* for details. Use this method as a quick first pass to get an idea of what constraints can be set.

If there are multiple timing exception constraints on the same object, the software uses the guidelines described in Conflict Resolution for Timing Exceptions, on page 197 to determine the constraint that takes precedence.

See Also

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

- Timing Constraints, on page 278
- *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 1911) on your sdc files.

Note: If you need to edit an SDC file, either use a text editor, or double-click the file to open the legacy SCOPE editor. For information on editing older SDC files, see SCOPE User Interface (Legacy), on page 201.

See Also

To use the current SCOPE editor, see:

- Chapter 5, *SCOPE Constraints Editor*
- Chapter 5, *Specifying Constraints*

Constraint Translation

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 135 Converting SDC to FDC, on page 191 in the *User Guide*

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 135
- sdc2fdc FPGA Design Constraint (FDC) File, on page 137
- sdc2fdc, on page 111 in the *Command Reference* manual (syntax)

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
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.

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 clkb:r
define_input_delay -disable {d[7:0]} 0.30 -ref clkb:r
define_output_delay -disable {x[7:0]} -improve 0.00 -route 0.00
define_output_delay -disable {y[7:0]} -improve 0.00 -route 0.00
#
#Registers
#
#
#Multicycle Path
#
#
#False Path
#
#
define_false_path -disable -from {i:x[1]}
#
```

```
#Path Delay
#
#
#Attributes
#
define_io_standard -default_input -delay_type input syn_pad_type {LVCMOS_33}#

#I/O standards
#
#
#Compile Points
#
#
#Other Constraints

#############################################################################
#SDC compliant constraints translated from Legacy Timing Constraints
#############################################################################
#
set_rtl_ff_names {#}

create_clock -name {clka} [get_ports {clka}] -period 10 -waveform {0 5.0}
create_clock -name {clkb} [get_ports {clkb}] -period 6.666666666666667
   -waveform {0 3.3333333333333335}
set_input_delay -clock [get_clocks {clka}] -clock_fall -
add_delay 0.000 [all_inputs]
set_output_delay -clock [get_clocks {clka}] -add_delay 0.000 [all_outputs]
set_input_delay -clock [get_clocks {clka}] -
add_delay 2.00 [get_ports {a[7:0]}]
set_input_delay -clock [get_clocks {clka}] -add_delay 0 [get_ports {rst}]
set mcp 4
set_multicycle_path $mcp -start  \
     -from \
       [get_ports \
       {a* \
       b*} \
       ] \
     -to \
       [find -seq -hier {q?[*]} ]

set_multicycle_path 3 -end  \
     -from \
       [find -seq {*y*.q[*]} ]

set_clock_groups -name default_clkgroup_0 -asynchronous \
    -group [get_clocks {clka dcm|clk0_derived_clock dcm|
   clk2x_derived_clock dcm|clk0fx_derived_clock}]
set_clock_groups -name default_clkgroup_1 -asynchronous \
    -group [get_clocks {clkb}]
```

Constraint Checking

The synthesis tools include several features to help you debug and analyze design constraints. Use the constraint checker to check the syntax and applicability of the timing constraints in the project. The synthesis log file includes a timing report as well as detailed reports on the compiler, mapper, and resource usage information for the design. A stand-alone timing analyzer (STA) generates a customized timing report when you need more details 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 174 for details.

Timing Constraint Report Files

The results of running constraint checking, synthesis, and stand-alone timing analysis are provided in reports that help you analyze constraints.

Use these files for additional timing constraint analysis:

File	Description
`_cck.rpt`	Lists the results of running the constraint checker (see *Constraint Checking Report* , on page 174).
`_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 36 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 383).
`.srr` or `.htm`	Reports post-synthesis timing results as part of the text or HTML log file (see *Timing Reports* , on page 165 and *Log File* , on page 160).

Database Object Search

To apply constraints, you have to search the database to find the appropriate objects. Sometimes you might want to search for and apply the same constraint to multiple objects. The FPGA tools provide some Tcl commands to facilitate the search for database objects:

Commands	**Common Commands**	**Description**
Find	Tcl Find, open_design…	Lets you search for design objects to form collections that can apply constraints to the group. See *Using Collections* , on page 181 and *find* , on page 151.
Collections	define_collection, c_union…	Create, copy, evaluate, traverse, and filter collections. See *Using Collections* , on page 181 and *Collection Commands* , on page 177 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
Lattice	*_SYNPLIFY.LPF SCF

For information about how forward annotation is handled for your target technology, refer to the appropriate vendor chapter of the *Reference Manual*.

Auto Constraints

Auto constraints are automatically generated by the synthesis tool, however, these do not replace regular timing constraints in the normal synthesis flow. Auto constraints are intended as a quick first pass to evaluate the kind of timing constraints you need to set in your design.

To enable this feature and automatically generate register-to-register constraints, use the Auto Constrain option on the left panel of the Project view. For details, see *Using Auto Constraints*, on page 393 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 146
- Libraries, on page 150
- Output Files, on page 154
- Log File, on page 160
- Timing Reports, on page 165
- Constraint Checking Report, on page 174

Input Files

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

Extension	File	Description
`.adc`	Analysis Design Constraint	Contains timing constraints to use for stand-alone timing analysis. Constraints in this file are used only for timing analysis and do not change the result files from synthesis. Constraints in the adc file are applied in addition to sdc constraints used during synthesis. Therefore, adc constraints affect timing results only if there are no conflicts with sdc constraints. You can forward annotate adc constraints to your vendor constraint file without rerunning synthesis. See *Using Analysis Design Constraints* , on page 386 of the *User Guide* for details.
`.fdc`	Synopsys FPGA Design Constraint	Create FPGA timing and design constraints with SCOPE. You can run the sdc2fdc utility to translate legacy FPGA timing constraints (SDC) to Synopsys FPGA timing constraints (FDC). For details, see the *sdc2fdc* , on page 111.
`.ini`	Configuration and Initialization	Governs the behavior of the synthesis tool. You normally do *not* need to edit this file. For example, use the HDL Analyst Options dialog box, instead, to customize behavior. See *HDL Analyst Options Command* , on page 477. On the Windows 7 platforms, the `ini` file is in the C:\Users\userName\AppData\Roaming\Synplicity directory, and on the Windows XP platforms, the `ini` file is in the C:\Documents and Settings\userName\Application Data\Synplicity directory. On Linux workstations, the `ini` file is in the following directory: (~/.Synplicity, where ~ is your home directory, which can be set with the environment variable $HOME).
`.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.

Extension	File	Description
`.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.
`.sv`	Source files (Verilog)	Design source files in SystemVerilog format. The `sv` source file is added to the Verilog directory in the Project view. For more information about the Verilog and SystemVerilog languages, and the synthesis commands and attributes you can include, see *Verilog* , on page 149, 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 58 of the *User Guide.*
`.vhd`	Source files (VHDL)	Design source files in VHDL format. See *VHDL* , on page 148 and Chapter 3, *VHDL Language Support* for details. For information about using VHDL and Verilog files together in a design, see *Using Mixed Language Source Files* , on page 58 of the *User Guide.*
`.v`	Source files (Verilog)	Design source files in Verilog format. For more information about the Verilog language, and the synthesis commands and attributes you can include, see *Verilog* , on page 149, 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 58 of the *User Guide.*

HDL Source Files

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*.

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 Chapter 16, *Optimizing for Lattice Designs*.

- 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*, on page 152.

- 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 151 and Open Verification Library, on page 152 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 RTL, 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 151.

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 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 RTL, 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.

Open Verification Library

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".
```

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
`_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 (*projectName*_cck.rpt). See *Constraint Checking Report* , on page 174 for more information.
`_compiler.linkerlog`	Compiler log file for HDL source file linking	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.
`.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.
`.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.

Extension	File	Description
.pfl	Message Filter criteria	Output file created after filtering messages in the Messages window. See *Updating the projectName.pfl file* , on page 243 in the *User Guide*.
Results file: • .edf • .edn	Vendor-specific results file	Results file that contains the synthesized netlist, written out in a format appropriate to the technology and the place-and-route tool you are using. Specify this file on the Implementation Results panel of the Implementation Options dialog box (*Implementation Results Panel* , on page 370).
run_options.txt	Project settings for implementations	This file is created when a design is synthesized and contains the project settings and options used with the implementations. These settings and options are also processed for displaying the Project Status view after synthesis is run. For details, see *Project Status Tab* , on page 31.
.sap	Synplify Annotated Properties	This file is generated after the Annotated Properties for Analyst option is selected in the Device panel of the Implementation Options dialog box. After the compile stage, the tool annotates the design with properties like clock pins. You can find objects based on these annotated properties using Tcl Find. For more information, see *find* , on page 151*Using the Tcl Find Command to Define Collections* , on page 176 in the *User Guide*.

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 353 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.
`.srr`	Synthesis log file	Provides information on the synthesis run, as well as area and timing reports. See *Log File* , on page 160, for more information.
`.srs`	Compiler output file	Output file after the compiler stage of the synthesis process. It contains an RTL-level representation of a design. This is the representation that appears graphically in an RTL view.

Extension	File	Description
`synlog folder`	Intermediate technology mapping files	This folder contains intermediate netlists and log files after technology mapping has been run. Timestamp information is contained in these netlist files to manage jobs with up-to-date checks. For more information, see *Using Up-to-date Checking for Job Management* , on page 214.
`synwork folder`	Intermediate pre-mapping files	This folder contains intermediate netlists and log files after pre-mapping has been run. Timestamp information is contained in these netlist files to manage jobs with up-to-date checks. For more information, see *Using Up-to-date Checking for Job Management* , on page 214.
`.ta`	Customized Timing Report	Contains the custom timing information that you specify through Analysis->Timing Analyst. See *Analysis Menu* , on page 440, for more information.
`_ta.srm`	Customized mapping output file	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 440 for more information.

Extension	File	Description
`.tap`	Timing Annotated Properties	This file is generated after the Annotated Properties for Analyst option is selected in the Device panel of the Implementation Options dialog box. After the compile stage, the tool annotates the design with timing properties and the information can be analyzed in the RTL view. 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 176 in the *User Guide*.
`.tlg`	Log file	This log file contains a list of all the modules compiled in the design.
v*endor constraint file*	Constraints file for forward annotation	Contains synthesis constraints to be forward-annotated to the place-and-route tool. The constraint file type varies with the vendor and the technology. Refer to the vendor chapters for specific information about the constraints you can forward-annotate. Check the Implementation Results dialog (Implementation Options) for supported files. See *Implementation Results Panel* , on page 370.

Extension	File	Description
.vm .vhm	Mapped Verilog or VHDL netlist	Optional post-synthesis netlist file in Verilog (.vm) or VHDL (.vhm) format. This is a structural netlist of the synthesized design, and differs from the original RTL used as input for synthesis. Specify these files on the Implementation Results dialog box (Implementation Options). See *Implementation Results Panel* , on page 370. Typically, you use this netlist for gate-level simulation, to verify your synthesis results. Some designers prefer to simulate before and after synthesis, and also after place-and-route. This approach helps them to isolate the stage of the design process where a problem occurred. The Verilog and VHDL output files are for functional simulation only. When you input stimulus into a simulator for functional simulation, use a cycle time for the stimulus of 1000 time ticks.

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 84) 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 161
High reliability report	*High Reliability Report* , on page 162
Buffers added to clocks in certain supported technologies.	*Clock Buffering Report* , on page 162
Buffers added to nets.	*Net Buffering Report* , on page 162
Compile point remapping	*Compile Point Information* , on page 163

For information about ...	See ...
Timing results. This section of the log file begins with "START TIMING REPORT" section. If you use the Timing Analyst to generate a custom timing report, its format is the same as the timing report in the log file, but the customized timing report is in a `ta` file.	*Timing Reports* , on page 165
Resources used by synthesis mapping	*Resource Usage Report* , on page 163
Design changes made as a result of retiming	*Retiming Report*, on page 164

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

Mapper Report

This report begins with the mapper version and date, and reports the following:

- Project information: the names of the constraint files, target technology, and attributes set in the design.

- Design information such as flattened instances, extraction of counters, FSM implementations, clock nets, buffered nets, replicated logic, RTL optimizations, and informational or warning messages.

High Reliability Report

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 31.

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

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 484of 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 165, for details.

You can use the Timing Analyst to generate additional timing reports for point-to-point analysis (see *Analysis Menu,* on page 440). 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 231 in the *User Guide* for a brief procedure on using the report to check for overutilization.

Retiming Report

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:

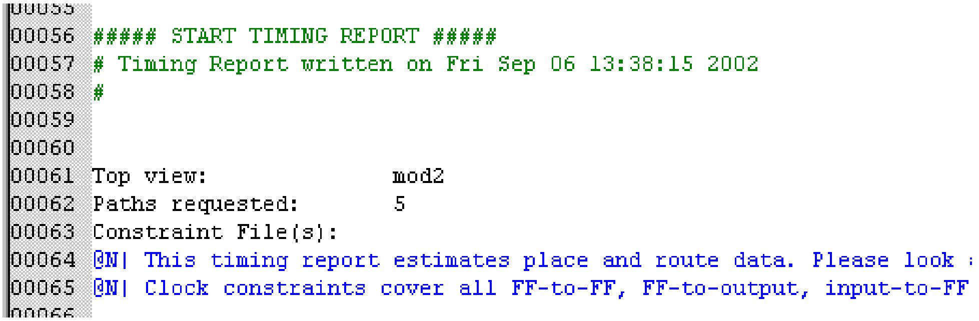
`.srr` or `.htm`	Log file that contains a default timing report. To find this information, after synthesis completes, open the log file (View -> Log File), and search for START OF TIMING REPORT.
`.ta`	Timing analysis file that contains timing information based on the parameters you specify in the stand-alone Timing Analyst (Analysis->Timing Analyst).
*designName*_async_clk .rpt.scv	Asynchronous clock report file that is generated when you enable the related option in the stand-alone Timing Analyzer (Analysis->Timing Analyst). This 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 172 for details on this report.

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

- Timing Report Header, on page 166
- Performance Summary, on page 166
- Clock Relationships, on page 168
- Interface Information, on page 170
- Detailed Clock Report, on page 170
- Asynchronous Clock Report, on page 172

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.



```
##### START TIMING REPORT #####
# Timing Report written on Fri Sep 06 13:38:15 2002
#


Top view:                mod2
Paths requested:         5
Constraint File(s):
@N| This timing report estimates place and route data. Please look
@N| Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF
```

You can control the size of the timing report by choosing Project -> Implementation Options, clicking the Timing Report tab of the panel, and specifying the number of start/end points and the number of critical paths to report. See *Timing Report Panel*, on page 371, for details.

Performance Summary

The Performance Summary section of the timing report reports estimated and requested frequencies for the clocks, with the clocks sorted by negative slack. The timing report has a different section for detailed clock information (see *Detailed Clock Report*, on page 170). The Performance Summary lists the following information for each clock in the design:

Performance Summary Column	Description
Starting Clock	Clock at the start point of the path. If the clock name is system, the clock is a collection of clocks with an undefined clock event. Rising and falling edge clocks are reported as one clock domain.

Performance Summary Column	Description
Requested/Estimated Frequency	Target frequency goal /estimated value after synthesis. See *Cross-Clock Path Timing Analysis* , on page 169 for information on how cross-clock path slack is reported.
Requested/Estimated Period	Target clock period/estimated value after synthesis.
Slack	Difference between estimated and requested period. See *Cross-Clock Path Timing Analysis* , on page 169 for information on how cross-clock path slack is reported.
Clock Type	The type of clock: inferred, declared, derived or system. For more information, see *Clock Types* , on page 167.
Clock Group	Name of the clock group that a clock belongs.

The synthesis tool does not report inferred clocks that have an unreasonable slack time. Also, a real clock might have a negative period. For example, suppose you have a clock going to a single flip-flop, which has a single path going to an output. If you specify an output delay of –1000 on this output, then the synthesis tool cannot calculate the clock frequency. It reports a negative period and no clock.

Clock Types

The synthesis timing reports include the following types of clocks:

- Declared Clocks

  User-defined clocks specified in the constraint file.

- Inferred Clocks

  These are clocks that the synthesis timing engine finds during synthesis, but which have not been constrained by the user. The tool assigns the default global frequency specified for the project to these clocks.

- Derived Clocks

  These are clocks that the synthesis tool identifies from a clock divider/multiplier such as DCM. The tool reports these clocks for timing purposes only.

- System Clock

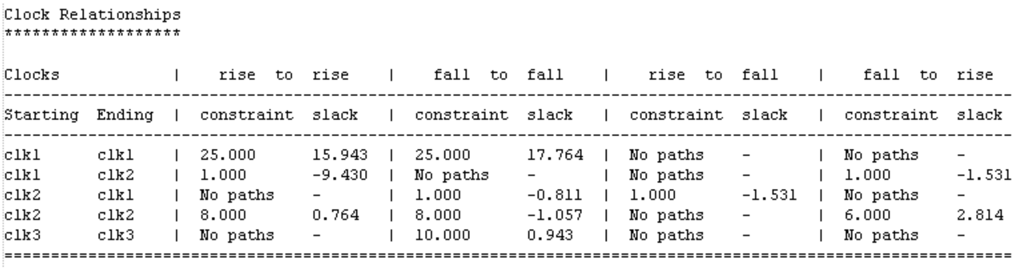
  The system clock is the delay for the combinational path. Additionally, a system clock can be reported if there are sequential elements in the design for a clock network that cannot be traced back to a clock. Also, the system clock can occur for unconstrained I/O ports. You must investigate these conditions.

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 169 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 157. For tips on using clock groups, see *Defining Other Clock Requirements*, on page 205 in the *User Guide*.



```
Clock Relationships
*******************

Clocks            |    rise  to  rise    |    fall  to  fall    |    rise  to  fall    |    fall  to  rise
-----------------------------------------------------------------------------------------------------------
Starting  Ending  |  constraint  slack   |  constraint  slack   |  constraint  slack   |  constraint  slack
-----------------------------------------------------------------------------------------------------------
clk1      clk1    |  25.000      15.943  |  25.000      17.764  |  No paths    -       |  No paths    -
clk1      clk2    |  1.000       -9.430  |  No paths    -       |  No paths    -       |  1.000       -1.531
clk2      clk1    |  No paths    -       |  1.000       -0.811  |  1.000       -1.531  |  No paths    -
clk2      clk2    |  8.000       0.764   |  8.000       -1.057  |  No paths    -       |  6.000       2.814
clk3      clk3    |  No paths    -       |  10.000      0.943   |  No paths    -       |  No paths    -
===========================================================================================================
```

Cross-Clock Path Timing Analysis

The following describe how the timing analyst calculates cross-clock path frequency and slack.

Cross-Clock Path Frequency

For each data path, the tool estimates the highest frequency that can be set for the clock(s) without a setup violation. It finds the largest scaling factor that can be applied to the clock(s) without causing a setup violation. If the start clock is not the same as the end clock, it scales both by the same factor.

scale = (*minimum time period* -(-*current slack*))/*minimum time period*

It assumes all other delays in the setup calculation (e.g., uncertainty) are fixed.

It applies relevant multicycle constraints to the setup calculation.

The estimated frequency for a clock is the minimum frequency over all paths that start or end on that clock, with the following exceptions:

- The tool does not consider paths between the system clock and another clock to estimate frequency.
- It considers paths with a path delay constraint to be asynchronous, and does not use them to estimate frequency.
- It considers paths between clocks in different domains to be asynchronous, and does not use them to estimate frequency.

Slack for Cross-Clock Paths

The slack reported for a cross-clock path is the worst slack for any path that starts on that clock. Note that this differs from the estimated frequency calculation, which is based on the worst slack for any path starting or ending on that clock.

Interface Information

The interface section of the timing report contains information on arrival times, required times, and slack for the top-level ports. It is divided into two subsections, one each for Input Ports and Output Ports. Bidirectional ports are listed under both. For each port, the interface report contains the following information.

Port parameter	**Description**
Port Name	Port name.
Starting Reference Clock	The reference clock.
User Constraint	The input/output delay. If a port has multiple delay records, the report contains the values for the record with the worst slack. The reference clock corresponds to the worst slack delay record.
Arrival Time	Input ports: define_input_delay, or default value of 0. Output ports: path delay (including clock-to-out delay of source register). For purely combinational paths, the propagation delay is calculated from the driving input port.
Required Time	Input ports: clock period – (path delay + setup time of receiving register + define_reg_input_delay value). Output ports: clock period – define_output_delay. Default value of define_output_delay is 0.
Slack	Required Time – Arrival Time

Detailed Clock Report

Each clock reported in the performance summary also has a detailed clock report section in the timing report. The clock reports are listed in order of negative slack.

General Critical Path Information

This section contains general information about the most critical paths in the design.

Clock Information	Description
N most critical start points	Start points can be input ports or registers. If the start point is a register, you see the starting pin in the report. To change the number of start points reported, choose Project -> Implementation Options, and set the number on the Timing Report panel.
N most critical end points	End points can be output ports or registers. If the end point is a register, you see the ending pin in the report. To change the number of end points reported, select Project -> Implementation Options, and set the number on the Timing Report panel.
N worst path information (see the next table for details)	Starting with the most critical path, the worst path Information sections contain details of the worst paths in the design. Paths from clock A to clock B are reported as critical paths in the section for clock A. You can change the number of critical paths on the Timing Report panel of the Implementation Options dialog box.

Worst Path Information

For each critical path, the timing report has a detailed description. It starts with a summary of the information and is followed by a detailed pin-by-pin report. The summary reports information like requested period, actual period, start and end points, and logic levels. Note that the requested period here is period -route delay, while the requested period in the Performance Summary (*Performance Summary*, on page 166) is just the clock period.

The detailed path report uses this format: Output pin – Net – Input pin – Output pin – Net – Input pin. The following table describes the critical path information reported:

Critical path information	Description
Instance/Net Name	Technology view names for the instances and nets in the critical path
Type	Type of cell
Pin Name	Name of the pin
Pin Dir	Pin direction

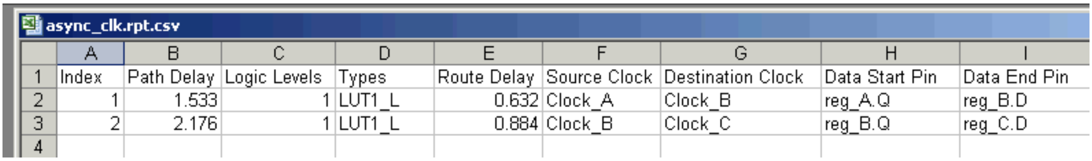
Critical path information	Description
Delay	The delay value.
Arrival Time	Clock delay at the source + the propagation delay through the path
Fan Out	Number of fanouts for the point in the path

Asynchronous Clock Report

You can generate a report for paths that cross between clock groups using the stand-alone Timing Analyst (Analysis->Timing Analyst, Generate Asynchronous Clock Report check box). Generally, paths in different clock groups are automatically handled as false paths. This option provides a file that contains information on each of the paths and can be viewed in a spreadsheet tool. To display the CSV-format report:

1. Locate the file in your results directory *projectName*_async_clk.rpt.csv.
2. Open the file in your spreadsheet tool.

Column	Description
Index	Path number.
Path Delay	Delay value as reported in standard timing (`ta`) file.
Logic Levels	Number of logic levels in the path (such as LUTs, cells, and so on) that are between the start and end points.
Types	Cell types, such as LUT, logic cell, and so on.
Route Delay	As reported for each path in `ta`
Source Clock	Start clock.
Destination Clock	End clock.
Data Start Pin	Sequential device output pin at start of path.
Data End Pin	Setup check pin at destination.



async_clk.rpt.csv

	A	B	C	D	E	F	G	H	I
1	Index	Path Delay	Logic Levels	Types	Route Delay	Source Clock	Destination Clock	Data Start Pin	Data End Pin
2	1	1.533	1	LUT1_L	0.632	Clock_A	Clock_B	reg_A.Q	reg_B.D
3	2	2.176	1	LUT1_L	0.884	Clock_B	Clock_C	reg_B.Q	reg_C.D
4									

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 63 in the *User Guide*.

This section describes the following topics:

- Reporting Details, on page 174
- Inapplicable Constraints, on page 175
- Applicable Constraints With Warnings, on page 176
- Sample Constraint Check Report, on page 177

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 175 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 176 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	• Ports • Nets • Pins • Registers • Instantiated buffers
create_generated_clock	Clocks
define_compile_point	• Region • View
define_current_design	v:*view*

For these constraints ...	Objects must be ...
set_false_path set_multicycle_path set_max_delay	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 set_reg_output_delay	Sequential instances

Applicable Constraints With Warnings

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

For these constraints ...	Objects must be ...
create_clock	• 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 set_multicycle_path set_path_delay	For -to or -from objects: • i:sequential instances • p:ports • i:black boxes For -through objects: • n:nets • t:hierarchical ports • t:pins
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 set_reg_output_delay	A single object. Multiple objects are not supported.

Sample Constraint Check Report

The following is a sample report generated by constraint checking:

```
# Synopsys Constraint Checker, version maprc, Build 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

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 590.

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 184
- Block RAM Examples, on page 195
- Initial Values for RAMs, on page 208
- RAM Instantiation with SYNCORE, on page 214
- ROM Inference, on page 215

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 Portable coding style Automatic timing-driven synthesis No additional tool dependencies	**Advantages** Most efficient use of the RAM primitives of a specific technology Supports all kinds of RAMs
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	**Limitations** 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	Lattice
Single Port	x
Dual Port	x
True Dual Port	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.

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 189 and *RAM Attributes*, on page 186. For inference examples, and see *Block RAM Examples*, on page 195.

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 196 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 197 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 198 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:

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

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

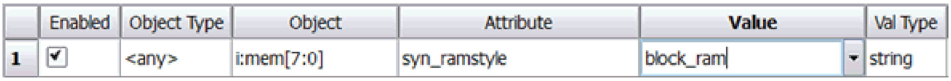
```
reg [7:0] ram_dout [127:0]
/*synthesis syn_ramstyle = "block_ram"*/;
reg [d_width-1:0] mem [mem_depth-1:0]
/*synthesis syn_rw_conflict_logic = 0*/;
```

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	☑	<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 189
- Simple Dual-Port Block RAM Inference, on page 191
- Dual-Port RAM Inference, on page 193
- True Dual-Port RAM Inference, on page 193
- True Dual-Port Byte-Enabled RAM Inference, on page 194

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:

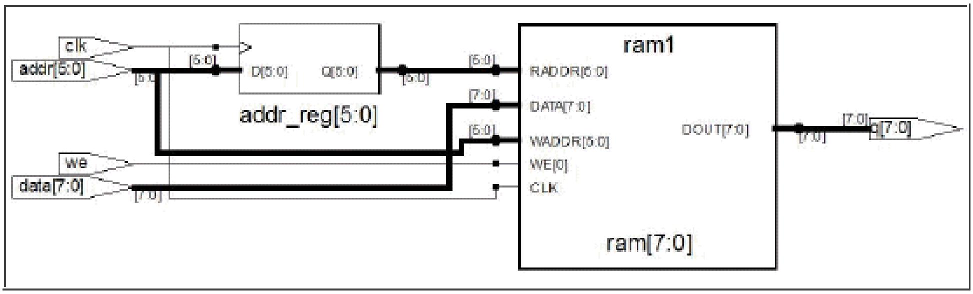
RAM	Clock	Read Ports	Write Ports
Single-port	Single clock	One; same as write	One; same as read
Simple dual-port	Single or dual clock	One dedicated read	One dedicated write
Dual-port	Single or dual clock	Two independent reads	One dedicated write
True dual-port	Single or dual clock	Two independent reads	Two independent writes
See *Dual-Port RAM Inference* , on page 193 and *True Dual-Port RAM Inference* , on page 193 for additional information.			

   For illustrative code examples, see the single-port and dual-port examples listed in *Block RAM Examples*, on page 195.

3. 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.

4. 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.

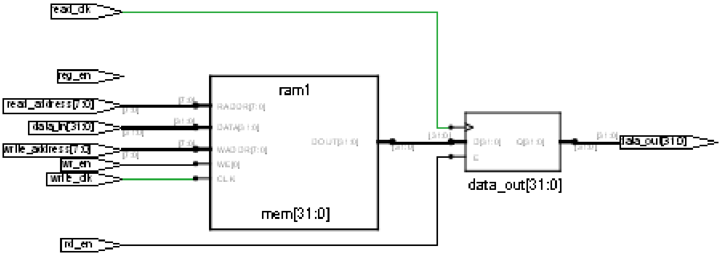
The synthesis tools map SDP RAMs to RAM primitives in the architecture. A unique set of addresses, clocks, and enable signals are used for each port. The synthesis tool might also set the RAM_MODE property on the RAM to indicate the RAM mode.

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.

1. 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 RTL code so that the synthesis tools can infer block RAM. See the following for details:

- Block RAM Mode Examples, on page 195
- Single-Port Block RAM Examples, on page 199
- Dual-Port Block RAM Examples, on page 202
- True Dual-Port RAM Examples, on page 204

For details about inferring block RAM, see *Block RAM Inference*, on page 189.

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:

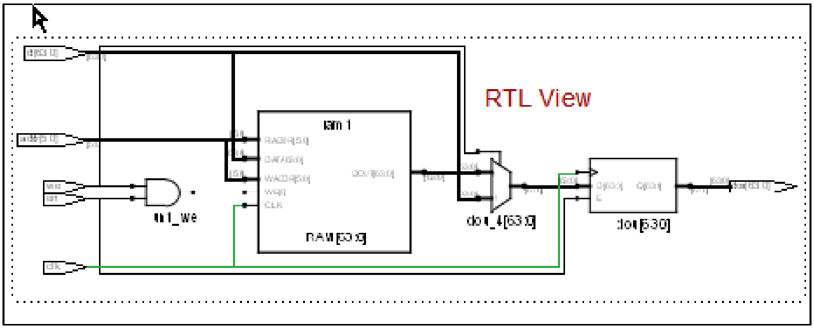
- WRITE_FIRST Mode Example, on page 196
- READ_FIRST Mode Example, on page 197
- NO_CHANGE Mode Example, on page 198

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
endmodule

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

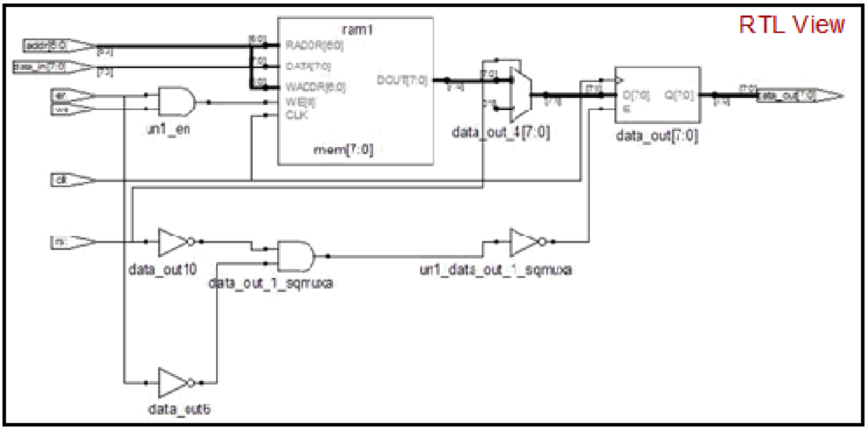


READ_FIRST Mode Example

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.

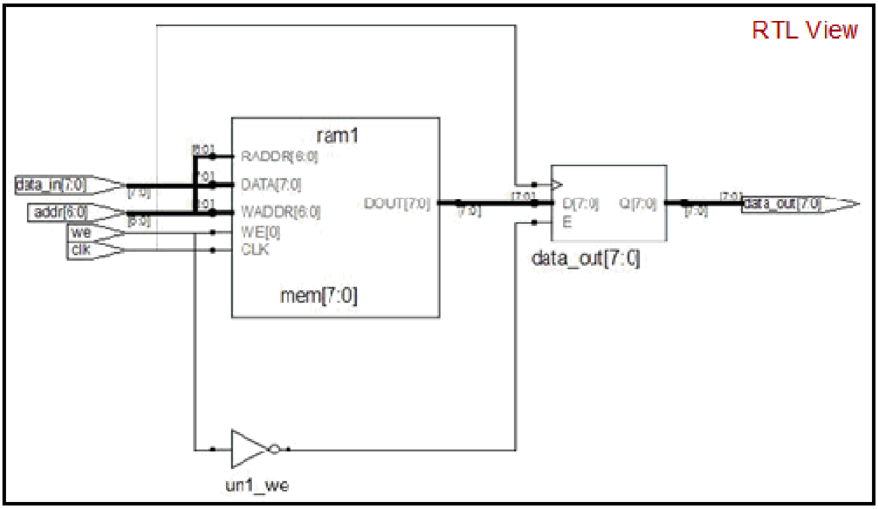


NO_CHANGE Mode Example

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 199
- Single-Port RAM with RAM Output Registered Examples, on page 201
- Dual-Port Block RAM Examples, on page 202

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 RTL 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:

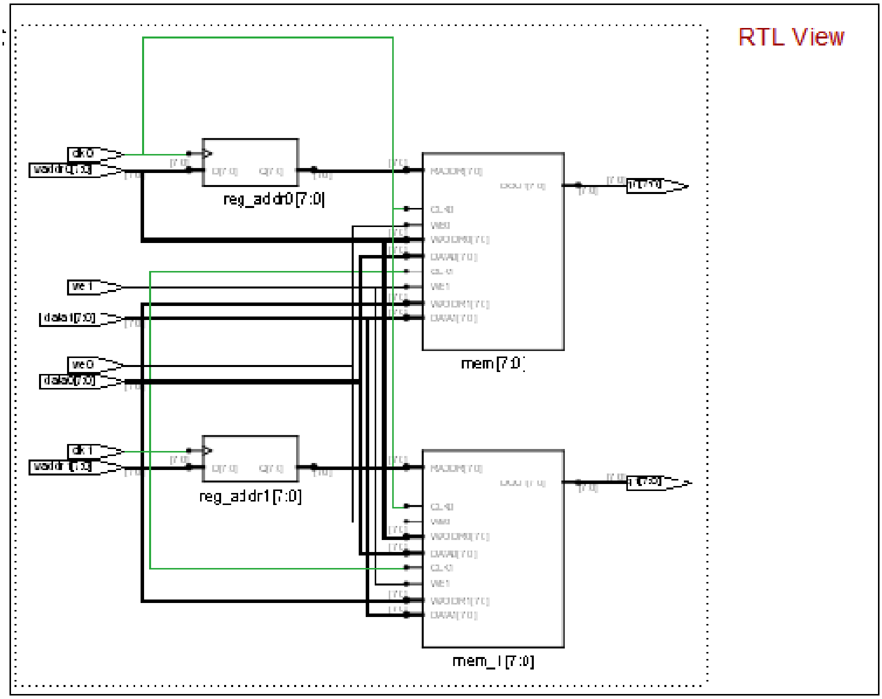
```
"@E:MF216: ram.v(29)|Found NRAM mem_1[7:0] with multiple
processes"
```

- 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 RTL 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 RTL 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
   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.

Initial Values for RAMs

You can specify initial values for a RAM in a data file and then include the appropriate task enable statement, $readmemb or $readmemh, in the initial statement of the RTL code for the module. The inferred logic can be different due to the initial statement. The syntax for these two statements is as follows:

$readmemh ("*fileName***",** *memoryName* **[,** *startAddress* **[,** *stopAddress***]]);**

$readmemb ("*fileName***",** *memoryName* **[,** *startAddress* **[,** *stopAddress***]]);**

$readmemb	Use this with a binary data file.
$readmemh	Use this with a hexadecimal data file.
fileName	Name of the data file that contains initial values. See *Initialization Data File* , on page 212 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 209
- Example 2: Cross-Module Referencing for RAM Initialization, on page 209
- Initialization Data File, on page 212
- Forward Annotation of Initial Values, on page 214

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 212 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.

```
// Example 2A: XMR for RAM Initialization
```

```
   (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
```

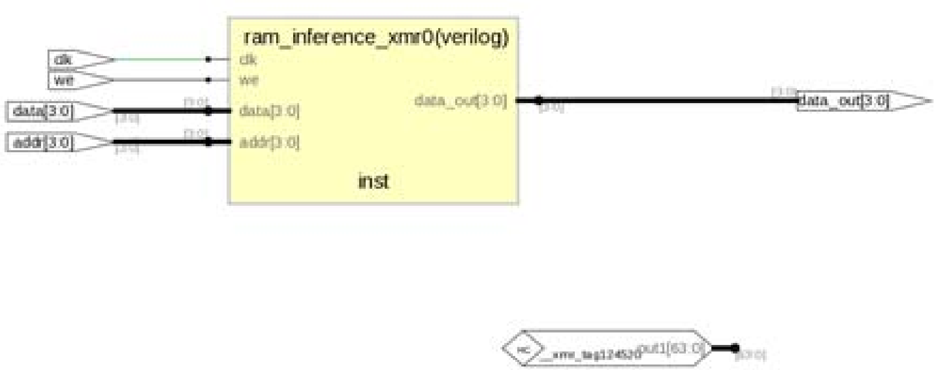
This code example implements cross-module referencing of the RAM block and is initialized with the $readmemb statement in the top-level module.

```
// Example 2B: 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 213), 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 213).

Binary File Format

The binary data file mem.ini that corresponds to the example in *Example 1: RAM Initialization*, on page 209 looks like this:

```
1111111111111111111100110111   /* data for address 0 */
1111111111111111111101100111   /* data for address 1 */
1111111111111111111111000010
1111111111111111111100100001
1111111111111111111101110000
1111111111111111111011100110
... /* 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 209 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.

RAM Instantiation with SYNCORE

The SYNCORE Memory Compiler in the IP Wizard helps you generate HDL code for your specific RAM implementation requirements. For information on using the SYNCORE Memory Compiler, see *Specifying RAMs with SYNCore,* on page 506 in the *User Guide*.

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

IP Tool

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

- SYNCore FIFO Compiler, on page 222
- SYNCore RAM Compiler, on page 240
- SYNCore Byte-Enable RAM Compiler, on page 250
- SYNCore ROM Compiler, on page 256
- SYNCore Adder/Subtractor Compiler, on page 262
- SYNCore Counter Compiler, on page 274

SYNCore FIFO Compiler

The SYNCore synchronous FIFO compiler offers an IP wizard that generates Verilog code for your FIFO implementation. This section describes the following:

- Synchronous FIFOs, on page 222
- FIFO Read and Write Operations, on page 223
- FIFO Ports, on page 225
- FIFO Parameters, on page 227
- FIFO Status Flags, on page 229
- FIFO Programmable Flags, on page 232

For further information, refer to the following:

- Specifying FIFOs with SYNCore, on page 500 of the *User Guide*, for information about using the wizard to generate FIFOs
- Launch SYNCore Command, on page 401 and SYNCore FIFO Wizard, on page 403 for descriptions of the interface

Synchronous FIFOs

A FIFO is a First-In-First-Out memory queue. Different control logic manages the read and write operations. A FIFO also has various handshake signals for interfacing with external user modules.

The SYNCore FIFO compiler generates synchronous FIFOs with symmetric ports and one clock controlling both the read and write operations. The FIFO is symmetric because the read and write ports have the same width.

When the Write_enable signal is active and the FIFO has empty locations, data is written into FIFO memory on the rising edge of the clock. A Full status flag indicates that the FIFO is full and that no more write operations can be performed. See *FIFO Write Operation, on page 223* 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 224* 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 corresponding read or write request to be ignored and the Overflow or Underflow flags to be asserted. You can monitor these status flags for invalid requests. These and other flags are described in *FIFO Status Flags*, on page 229 and *FIFO Programmable Flags*, on page 232.

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

FIFO Read and Write Operations

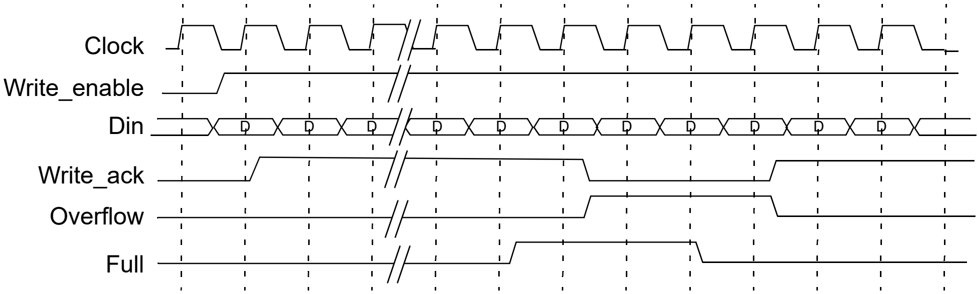
This section describes FIFO behavior with read and write operations.

FIFO Write Operation

When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus (Din) and write acknowledge (Write_ack) is asserted. If the FIFO is continuously written without being read, it will fill with data. The status outputs are asserted when the number of entries in the FIFO is greater than or equal to the corresponding threshold, and should be monitored to avoid overflowing the FIFO.

When the FIFO is full, any attempted write operation fails and the overflow flag is asserted.

The following figure illustrates the write operation. Write acknowledge (Write_ack) is asserted on the next rising clock edge after a valid write operation. When Full is asserted, there can be no more legal write operations. This example shows that asserting Write_enable when Full is high causes the assertion of Overflow.

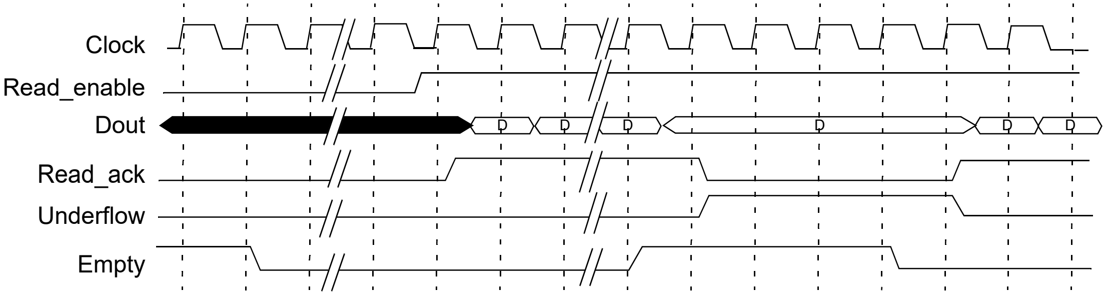


FIFO Read Operation

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.

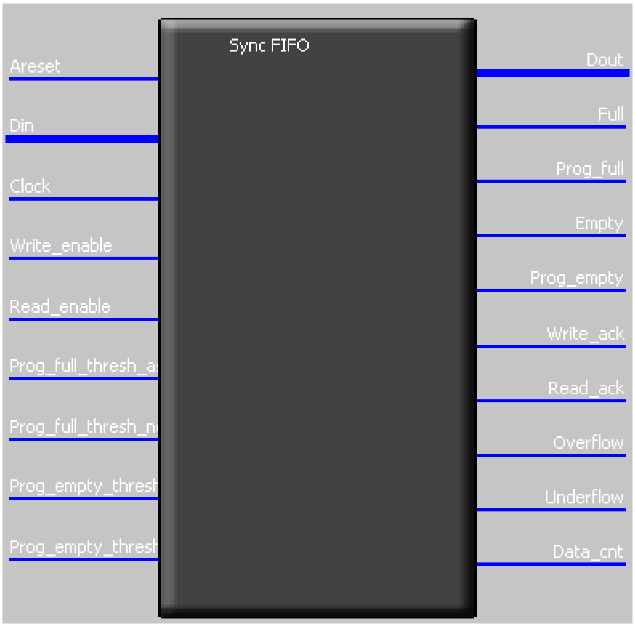
If read and write operation occur simultaneously during the empty state, the write operation will be valid and empty, and is de-asserted at the next rising clock edge. There cannot be a legal read operation from an empty FIFO, so the underflow flag is asserted.

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.

Port Name	Description
Dout [width:0]	Data output word from the FIFO.
Empty	FIFO empty output (active high). Asserted when the FIFO is empty and no additional reads can be performed. Can be active high or active low.
Full	FIFO full output (active high). Asserted when the FIFO is full and no additional writes can be performed. Can be active high or active low.
Overflow	FIFO overflow output flag (active high). Asserted when the FIFO is full and the previous write was rejected. Can be active high or active low.
Prog_empty	Programmable empty output flag (active high). Asserted when the words in the FIFO exceed or equal the programmable empty assert threshold. De-asserted when the number of words is more than the programmable full negate threshold. Can be active high or active low.
Prog_empty_ thresh	Programmable FIFO empty threshold input. User-programmable threshold value for the assertion of the Prog_empty flag. Set during reset.
Prog_empty_ thresh_assert	Programmable FIFO empty threshold assert input. User-programmable threshold value for the assertion of the Prog_empty flag. Set during reset.
Prog_empty_ thresh_negate	Programmable FIFO empty threshold negate input. User programmable threshold value for the de-assertion of the Prog_full flag. Set during reset.
Prog_full	Programmable full output flag (active high). Asserted when the words in the FIFO exceed or equal the programmable full assert threshold. De-asserted when the number of words is less than the programmable full negate threshold. Can be active high or active low.
Prog_full_thresh	Programmable FIFO full threshold input. User-programmable threshold value for the assertion of the Prog_full flag. Set during reset.
Prog_full_thresh_ assert	Programmable FIFO full threshold assert input. User-programmable threshold value for the assertion of the Prog_full flag. Set during reset.

Port Name	Description
Prog_full_thresh_negate	Programmable FIFO full threshold negate input. User-programmable threshold value for the de-assertion of the Prog_full flag. Set during reset.
Read_ack	Read acknowledge output (active high). Asserted when valid data is read from the FIFO. Can be active high or active low.
Read_enable	Read enable output (active high). If the FIFO is not empty, data is read from the FIFO on the next rising edge of the read clock.
Underflow	FIFO underflow output flag (active high). Asserted when the FIFO is empty and the previous read was rejected.
Write_ack	Write Acknowledge output (active high). Asserted when there is a valid write into the FIFO. Can be active high or active low.
Write_enable	Write enable input (active high). If the FIFO is not full, data is written into the FIFO on the next rising edge.

FIFO Parameters

Parameter	Description
AEMPTY_FLAG_SENSE	FIFO almost empty flag sense 0 Active Low 1 Active High
AFULL_FLAG_SENSE	FIFO almost full flag sense 0 Active Low 1 Active High
DEPTH	FIFO depth
EMPTY_FLAG_SENSE	FIFO empty flag sense 0 Active Low 1 Active High
FULL_FLAG_SENSE	FIFO full flag sense 0 Active LowOVERFLOW_ 1 Active High

Parameter	Description
OVERFLOW_FLAG_ SENSE	FIFO overflow flag sense 0 Active Low 1 Active High
PEMPTY_FLAG_ SENSE	FIFO programmable empty flag sense 0 Active Low 1 Active High
PFULL_FLAG_SENSE	FIFO programmable full flag sense 0 Active Low 1 Active High
PGM_EMPTY_ ATHRESH	Programmable empty assert threshold for PGM_EMPTY_TYPE=2
PGM_EMPTY_ NTHRESH	Programmable empty negate threshold for PGM_EMPTY_TYPE=2
PGM_EMPTY_THRESH	Programmable empty threshold for PGM_EMPTY_TYPE=1
PGM_EMPTY_TYPE	Programmable empty type. See *Programmable Empty* , on page 236 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 233 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

Parameter	Description
RACK_FLAG_SENSE	FIFO read acknowledge flag sense 0 Active Low 1 Active High
UNDERFLOW_FLAG_SENSE	FIFO underflow flag sense 0 Active Low 1 Active High
WACK_FLAG_SENSE	FIFO write acknowledge flag sense 0 Active Low 1 Active High
WIDTH	FIFO data input and data output width

FIFO Status Flags

You can set the following status flags for FIFO read and write operations.

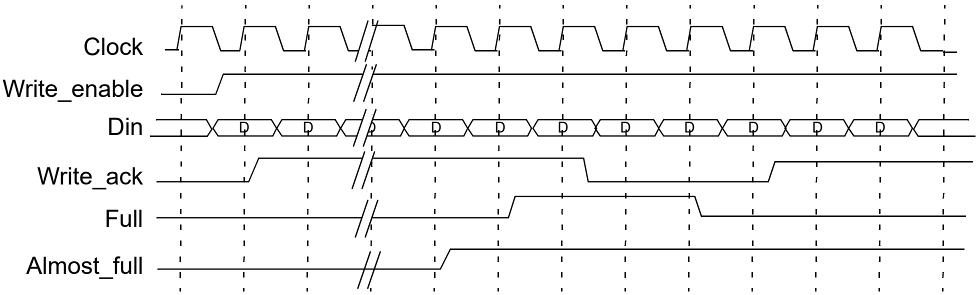
- Full/Almost Full Flags, on page 229
- Empty/Almost Empty Flags, on page 230
- Handshaking Flags, on page 231
- Programmable full and empty flags, which are described in *Programmable Full*, on page 233 and *Programmable Empty*, on page 236.

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 Wriite_enable when Almost_full is high causes the assertion of Full on the next rising clock edge.



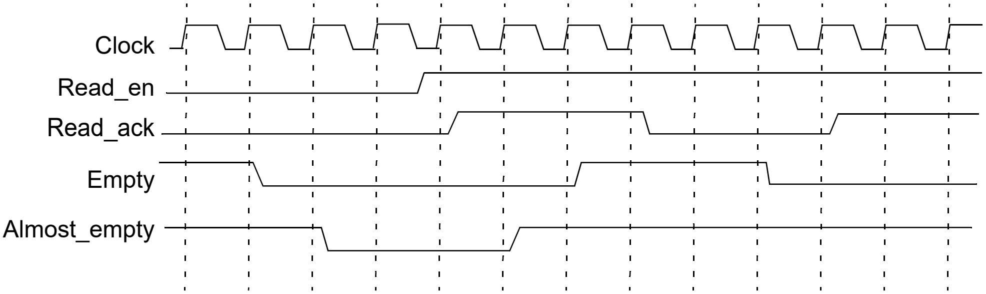
Empty/Almost Empty Flags

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

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

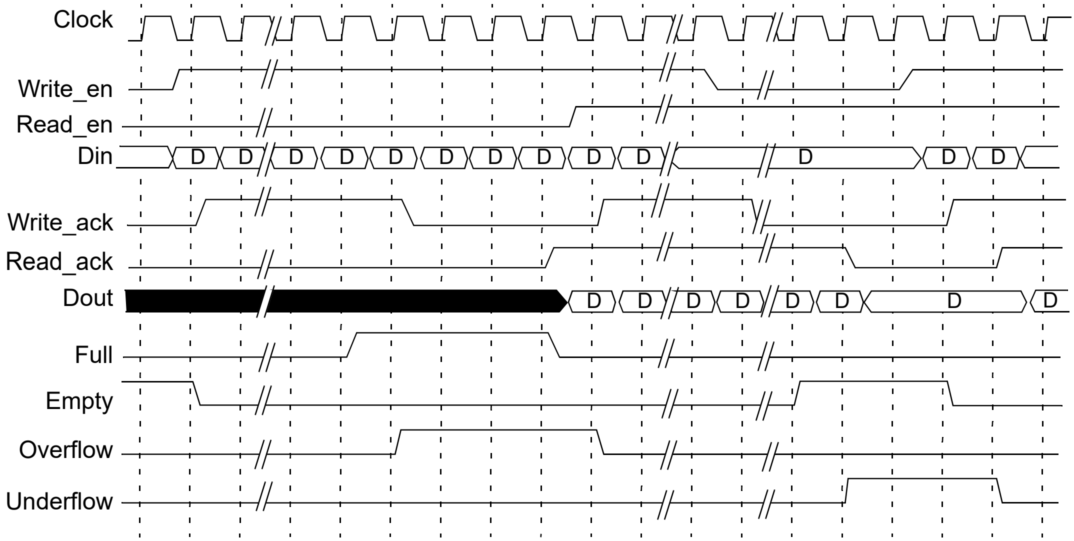
Almost_ empty — Indicates that the FIFO will be empty after one more read operation. Almost_empty is active high and is synchronous to the clock. The flag is guaranteed to be asserted when the FIFO has one remaining location for a read operation.

The following figure illustrates the behavior of the FIFO with one word remaining.



Handshaking Flags

You can specify optional Read_ack, Write_ack, Overflow, and Underflow handshaking flags for the FIFO.



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 233 for more information.

Prog_empty

Indicates that the FIFO has reached a user-defined empty threshold. See *Programmable Empty* , on page 236 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

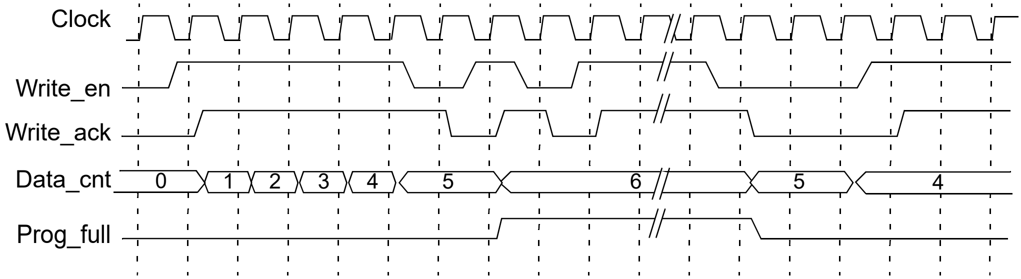
Prog_full has four threshold types:

- Programmable Full with Single Threshold Constant, on page 233
- Programmable Full with Multiple Threshold Constants, on page 234
- Programmable Full with Single Threshold Input, on page 234
- Programmable Full with Multiple Threshold Inputs, on page 235

Programmable Full with Single Threshold Constant

PGM_FULL_TYPE = 1

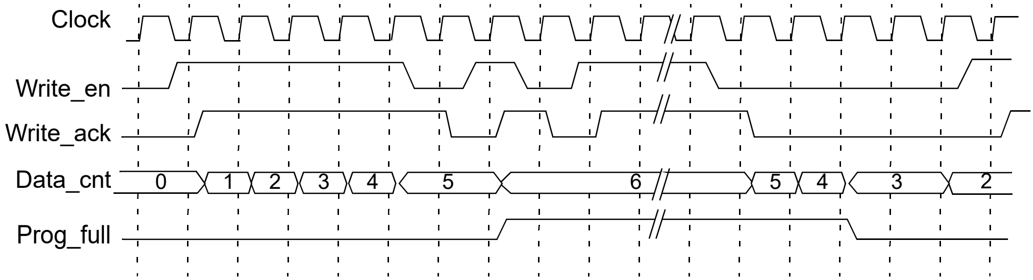
This option lets you set a single constant value for the threshold. It requires significantly fewer resources when the FIFO is generated. This figure illustrates the behavior of Prog_full when configured as a single threshold constant with a value of 6.



Programmable Full with Multiple Threshold Constants

PGM_FULL_TYPE = 2

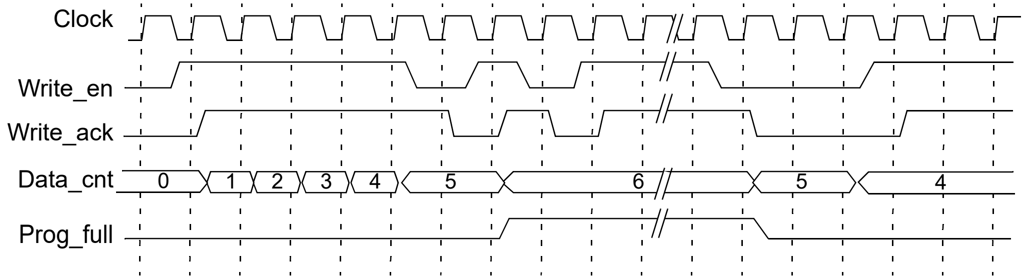
The programmable full flag is asserted when the number of words in the FIFO is greater than or equal to the full threshold assert value. If the number of FIFO words drops to less than the full threshold negate value, the programmable full flag is de-asserted. Note that the negate value must be set to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

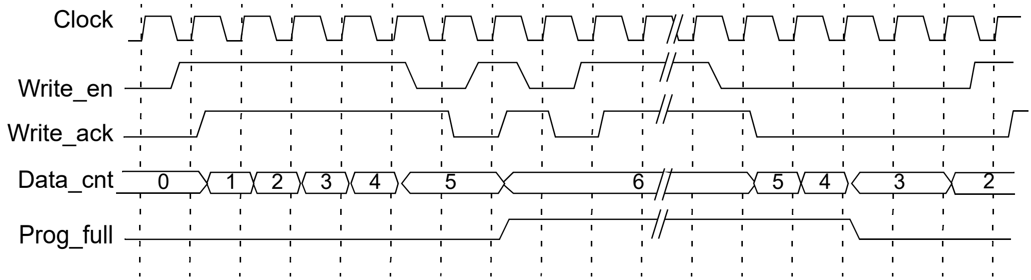


Programmable Full with Multiple Threshold Inputs

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

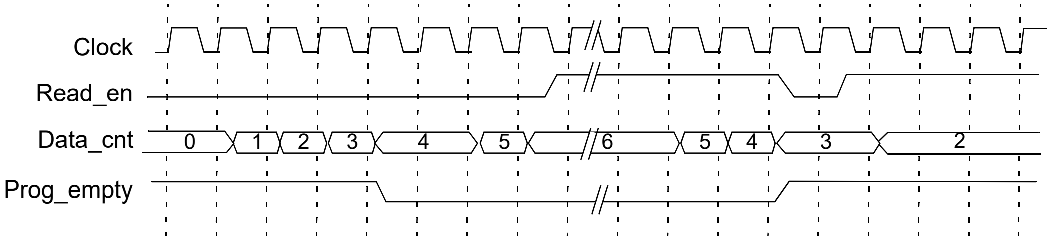
There are four threshold types you can specify:

- Programmable Empty with Single Threshold Constant, on page 236
- Programmable Empty with Multiple Threshold Constants, on page 237
- Programmable Empty with Single Threshold Input, on page 237
- Programmable Empty with Multiple Threshold Inputs, on page 239

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.

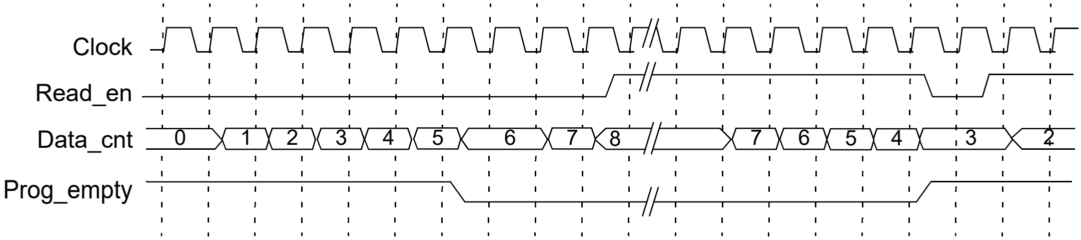


Programmable Empty with Multiple Threshold Constants

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.

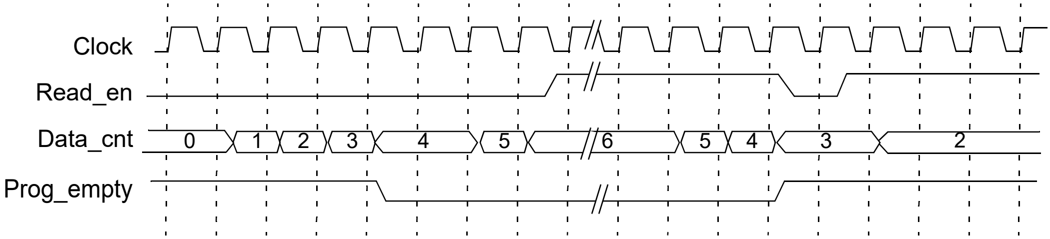
The following figure illustrates the behavior of Prog_empty when configured as multiple threshold constants with an assert value of 3 and a negate value of 5.



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.

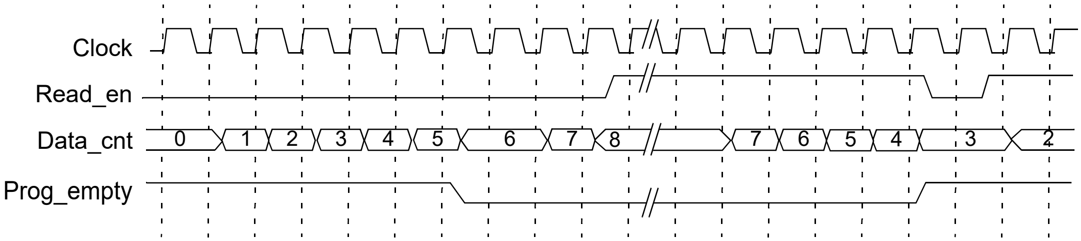

Clock
Read_en
Data_cnt
Prog_empty
0
1
2
3
4
5
6
5
4
3
2

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:

- Single-Port Memories, on page 240
- Dual-Port Memories, on page 242
- Read/Write Timing Sequences, on page 247

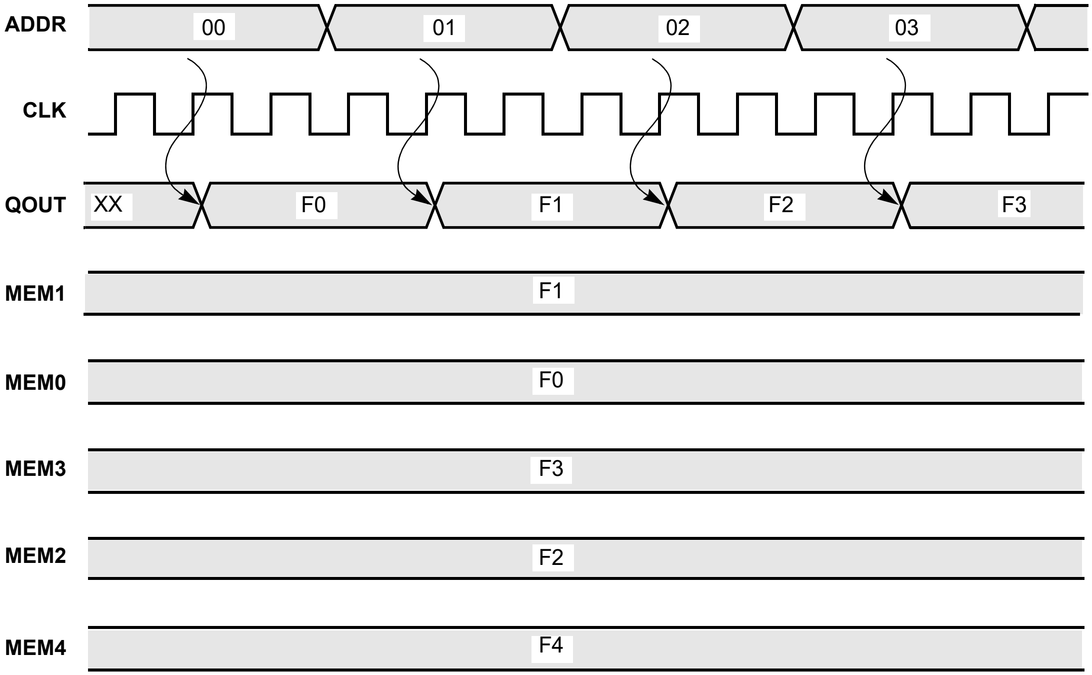
For further information, refer to the following:

- Specifying RAMs with SYNCore, on page 506 of the *User Guide*, for information about using the wizard to generate FIFOs
- Launch SYNCore Command, on page 401 and SYNCore FIFO Wizard, on page 403 for descriptions of the interface

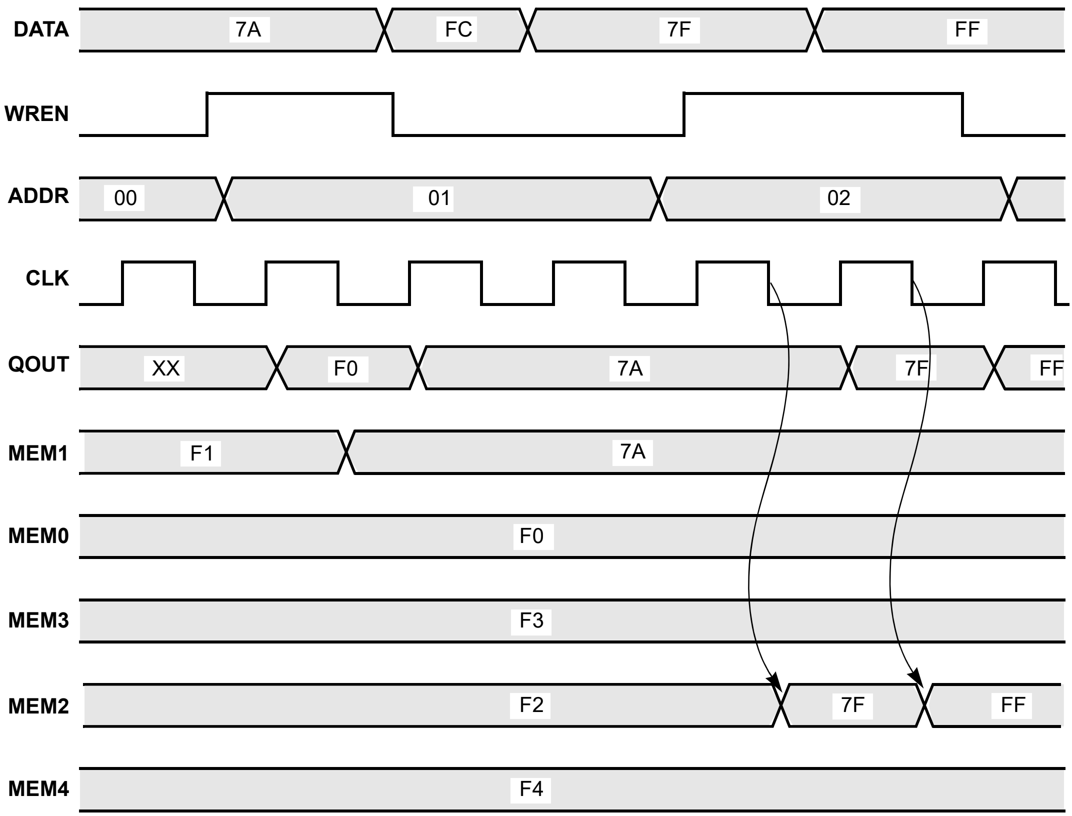
Single-Port Memories

For single-port RAM, it is only necessary to configure Port A. The following diagrams show the read-write timing for single-port memories. See *Specifying RAMs with SYNCore*, on page 506 in the *User Guide* for a procedure.

Single-Port Read



Single-Port Write



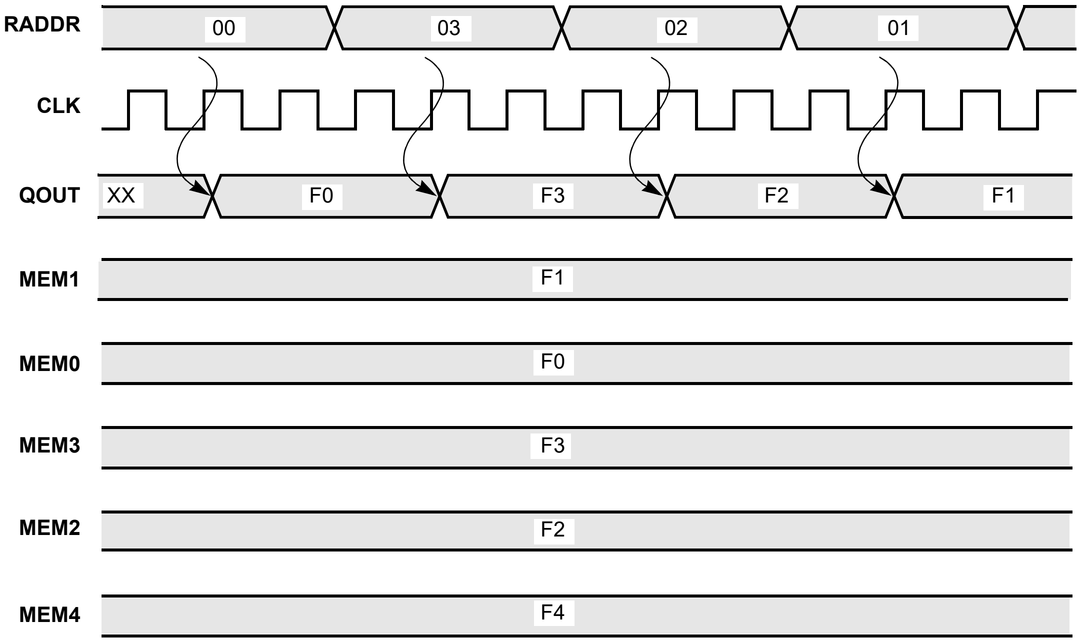
Dual-Port Memories

SYNCore dual-port memory includes the following common configurations:

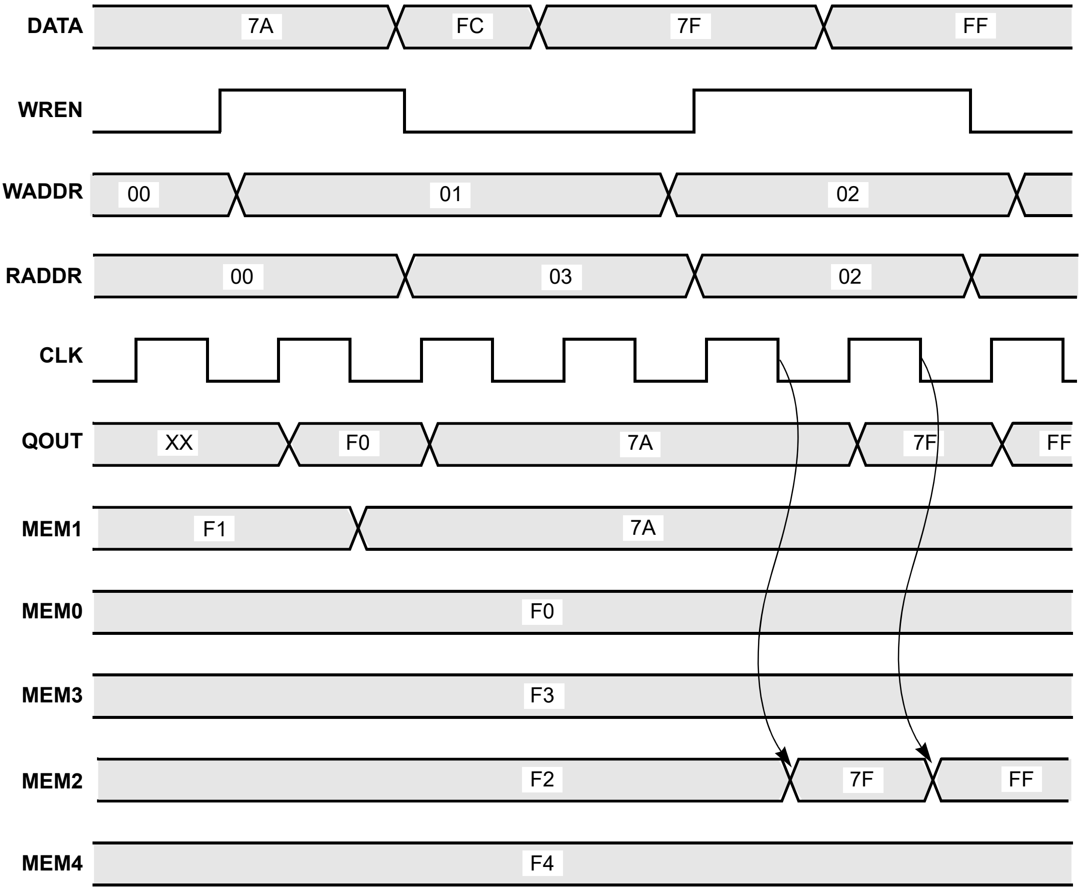
- One read access and one write access
- Two read accesses and one write access
- Two read accesses and two write accesses

The following diagrams show the read-write timing for dual-port memories. See *Specifying RAMs with SYNCore,* on page 506 in the *User Guide* for a procedure to specify a dual-port RAM with SYNCore.

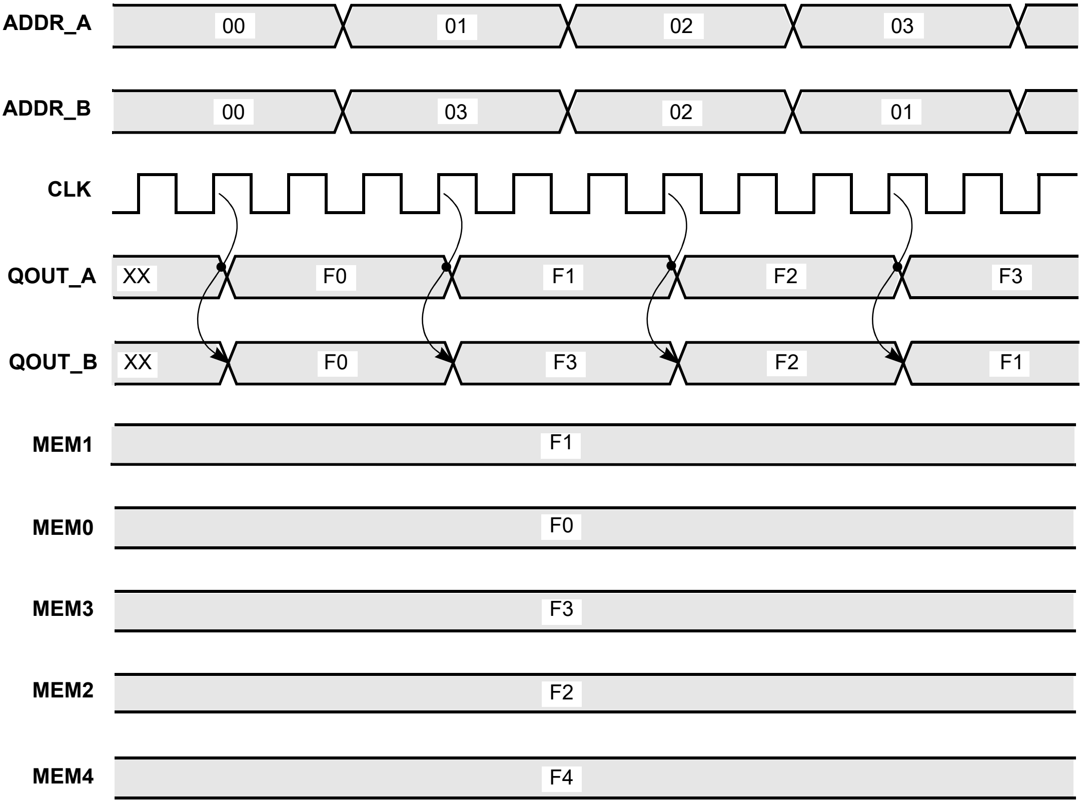
Dual-Port Single Read



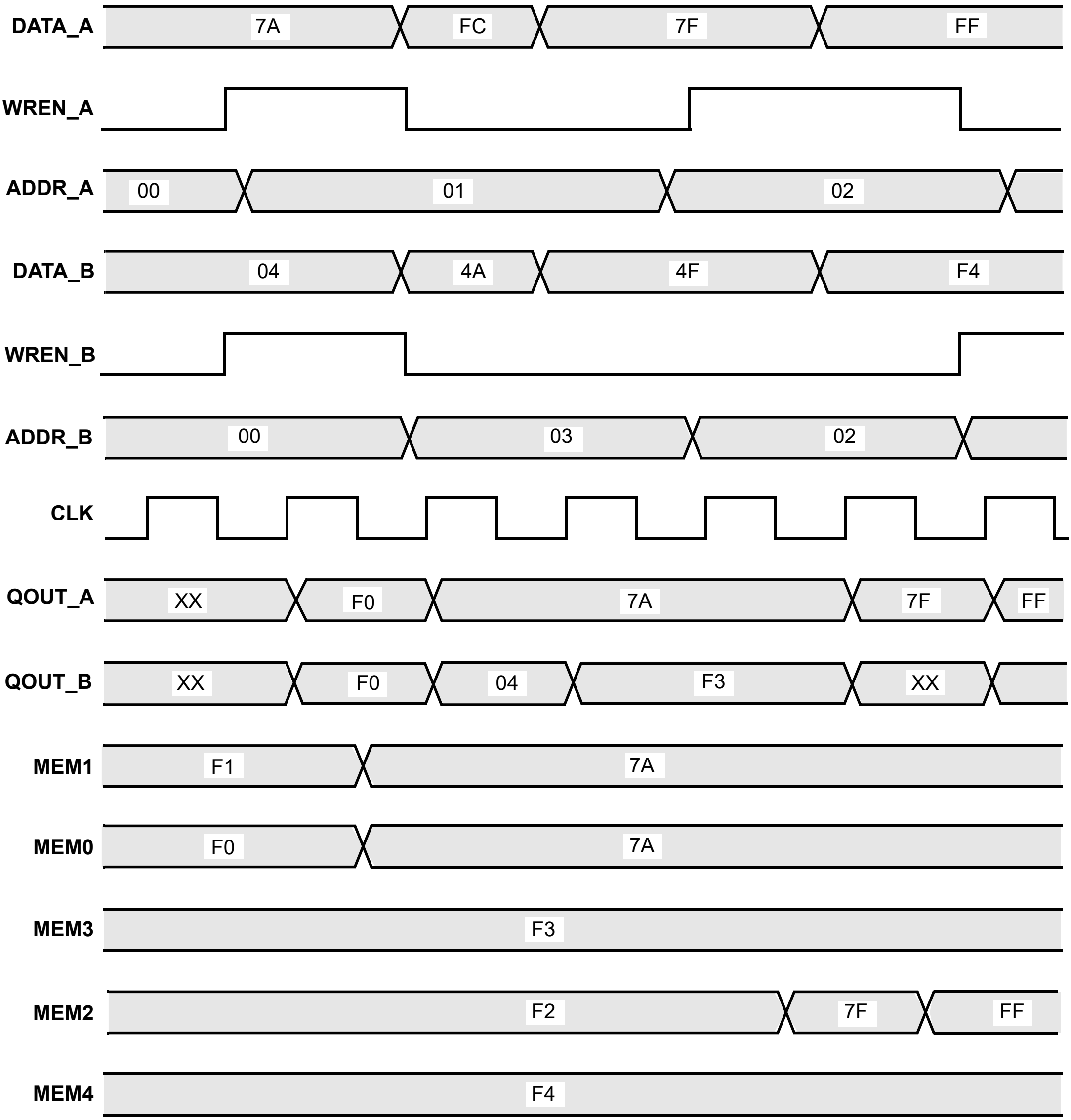
Dual-Port Single Write



Dual-Port Read



Dual-Port Write

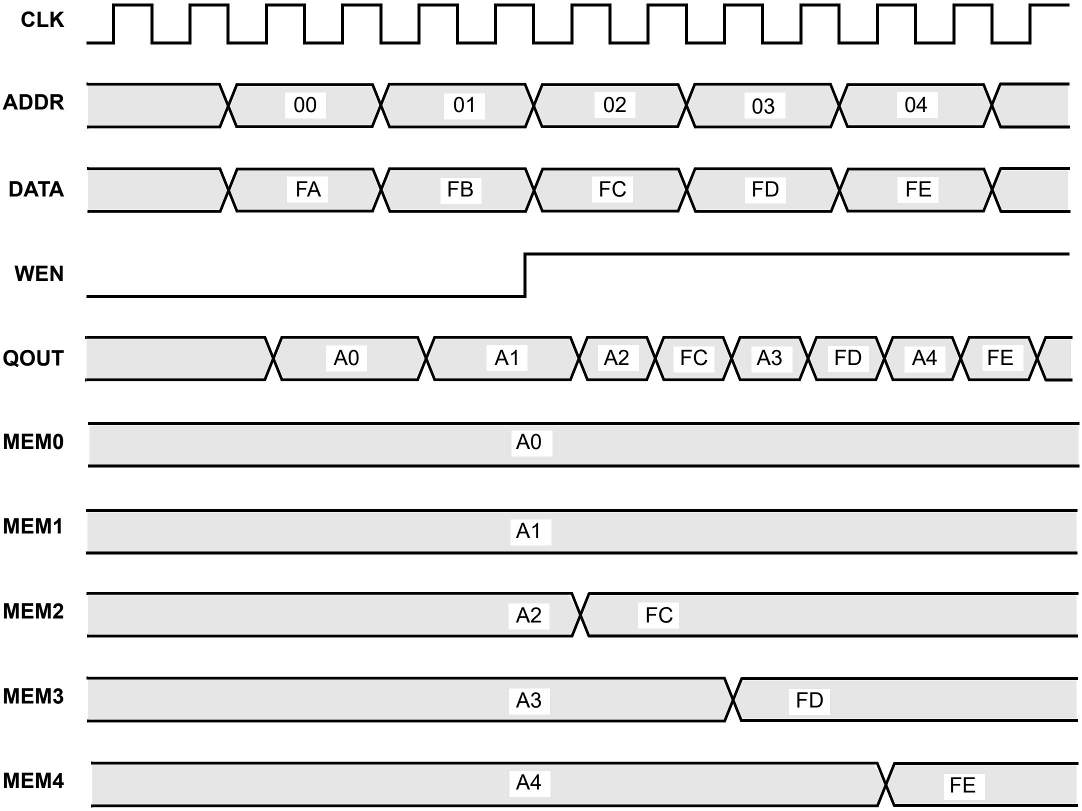


Read/Write Timing Sequences

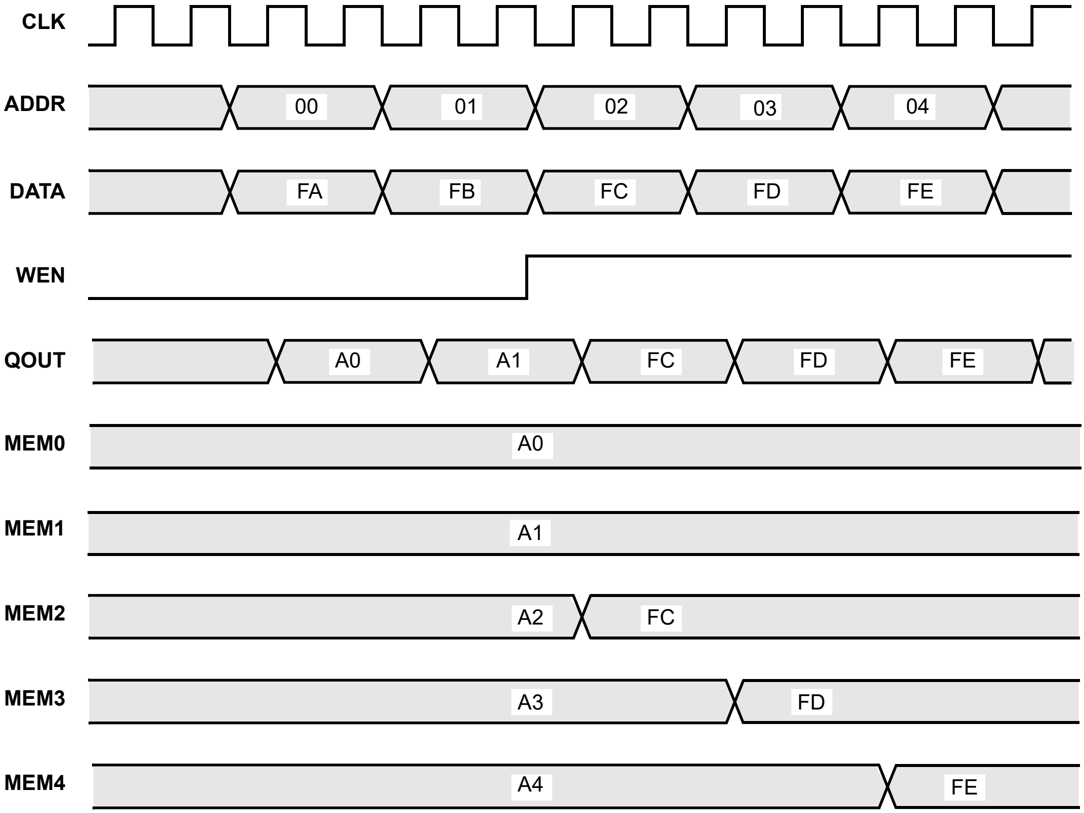
The waveforms in this section describe the behavior of the RAM when both read and write are enabled and the address is the same operation. The waveforms show the behavior when each of the read-write sequences is enabled. The waveforms are merged with the simple waveforms shown in the previous sections. See the following:

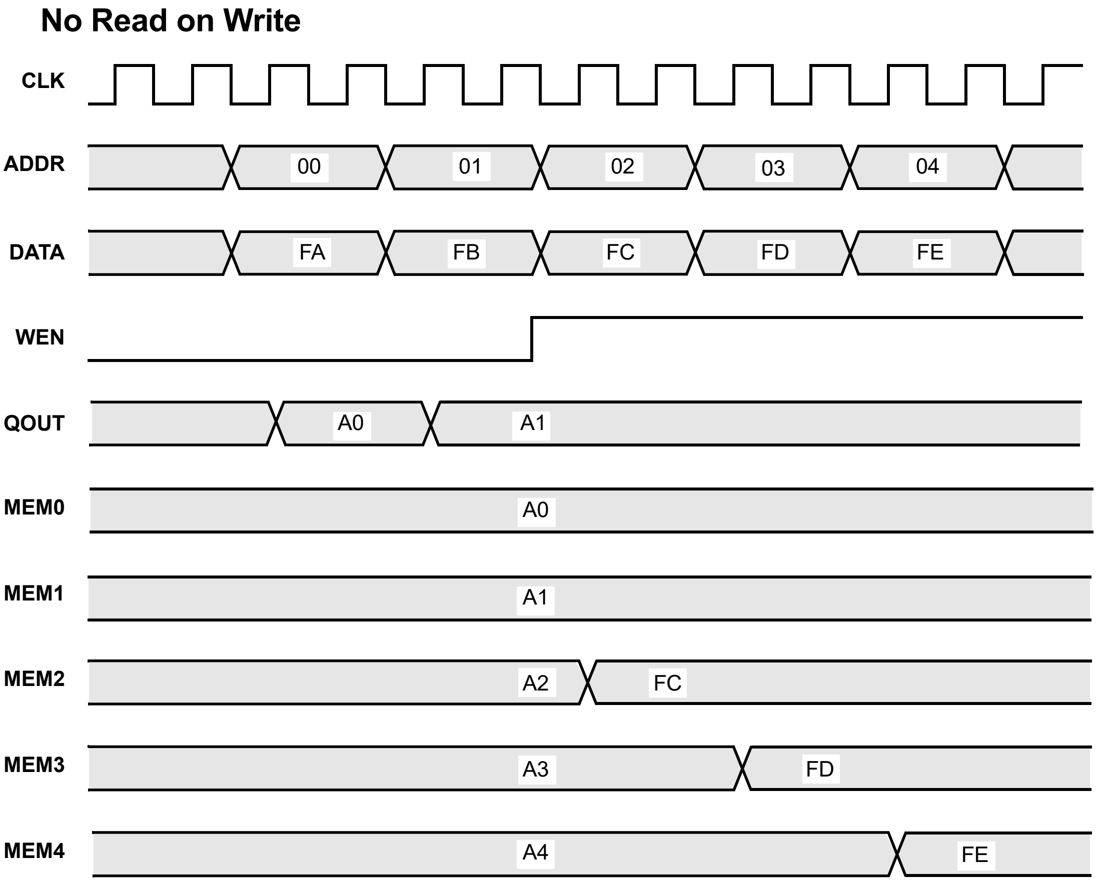
- Read Before Write, on page 247
- Write Before Read, on page 248
- No Read on Write, on page 249

Read Before Write



Write Before Read




No Read on Write
CLK
ADDR
00
01
02
03
04
DATA
FA
FB
FC
FD
FE
WEN
QOUT
A0
A1
MEM0
A0
MEM1
A1
MEM2
A2
FC
MEM3
A3
FD
MEM4
A4
FE

SYNCore Byte-Enable RAM Compiler

The SYNCore byte-enable RAM compiler generates SystemVerilog code describing byte-enabled RAMs. The data width of each byte is calculated by dividing the total data width by the write enable width. The byte-enable RAM compiler supports both single- and dual-port configurations.

This section describes the following:

- Functional Overview, on page
- Write Operation, on page
- Read Operation, on page
- Parameter List, on page

For further information, refer to the following:

- *Specifying Byte-Enable RAMs with SYNCore*, on page 513 of the user guide for information on using the wizard to generate single- or dual-port RAM configurations.
- *SYNCore Byte-Enable RAM Wizard*, on page 416 for descriptions of the interface.

Functional Overview

The SYNCore byte-enable RAM component supports bit/byte-enable RAM implementations using blockRAM and distributed memory. For each configuration, design optimizations are made for optimum use of core resources. The timing diagram that follow illustrate the supported signals for byte-enable RAM configurations.

Byte-enable RAM can be configured in both single- and dual-port configurations. In the dual-port configuration, each port is controlled by different clock, enable, and control signals. User configuration controls include selecting the enable level, reset type, and register type for the read data outputs and address inputs.

Reset applies only to the output read data registers; default value of read data on reset can be changed by user while generating core. Reset option is inactive when output read data is not registered.

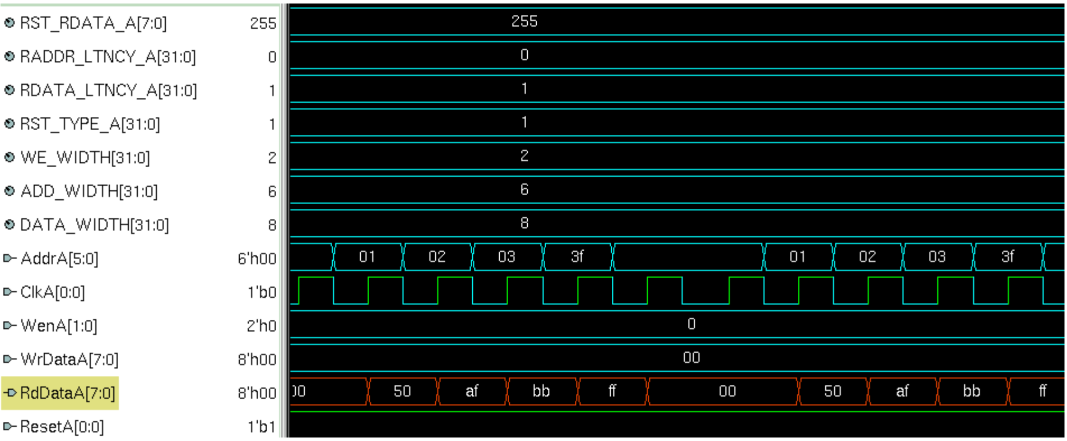
Read/Write Timing Sequences

The waveforms in this section describe the behavior of the byte-enable RAM for both read and write operations.

Read Operation

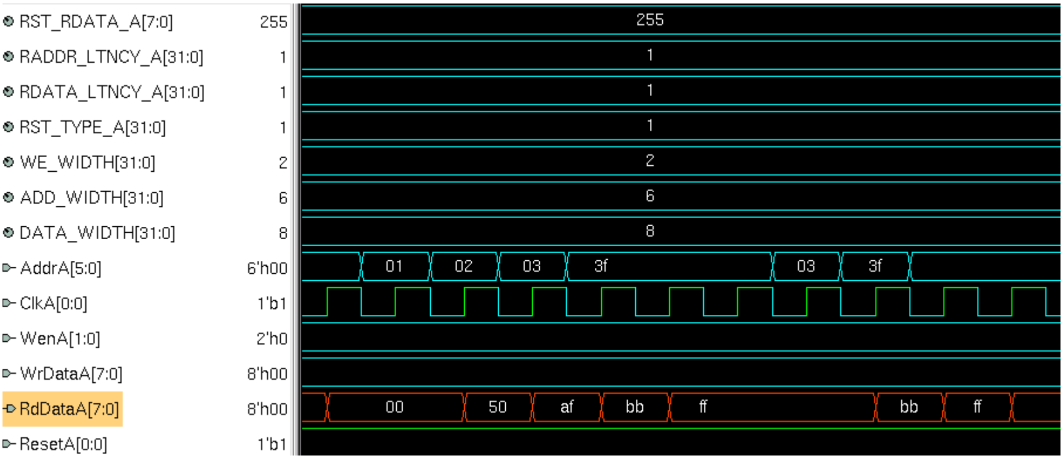
On each active edge of the clock when there is a change in address, data is valid on the same clock or next clock (depending on latency parameter values for read address and read data ports). Active reset ignores any change in input address, and data and output data are initialized to user-defined values 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.

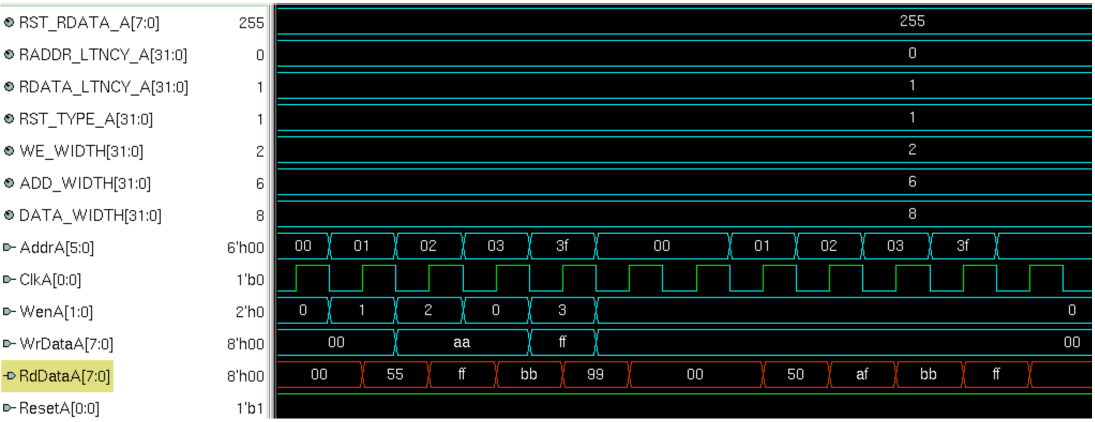


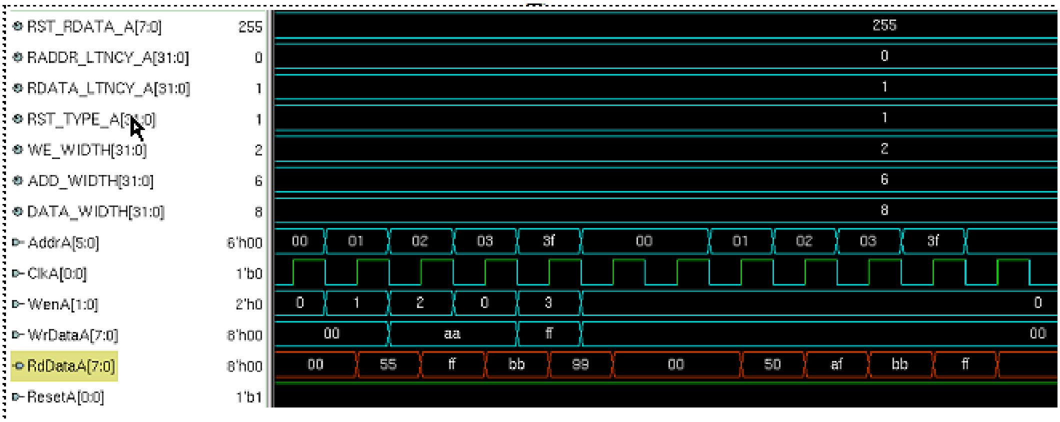
As shown in the above waveform, output read data changes on the next clock edge after the input address changes. When the address changes from 'h00 to 'h01, read data changes to 50 on the next clock, and data is valid on the next clock edge.

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:

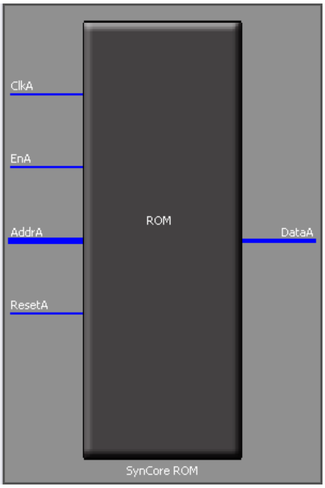
- Functional Overview, on page 256
- Single-Port Read Operation, on page 257
- Dual-Port Read Operation, on page 258
- Parameter List, on page 259
- Clock Latency, on page 260

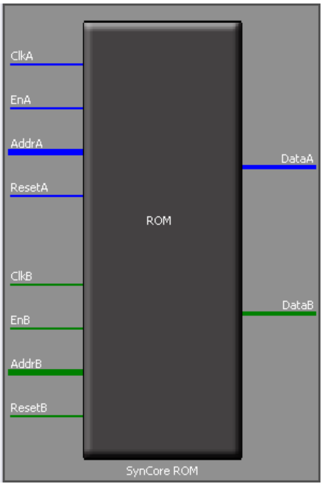
For further information, refer to the following:

- *Specifying ROMs with SYNCore,* on page 520 of the *User Guide*, for information about using the wizard to generate ROMs
- *Launch SYNCore Command,* on page 401 and *SYNCore ROM Wizard,* on page 419 for descriptions of the interface

Functional Overview

The SYNCore ROM component supports ROM implementations using block ROM or logic memory. For each configuration, design optimizations are made for optimum usage of core resources. Both single- and dual-port memory configurations are supported. Single-port ROM allows read access to memory through a single port, and dual-port ROM allows read access to memory through two ports. The following figure illustrates the supported signals for both configurations.





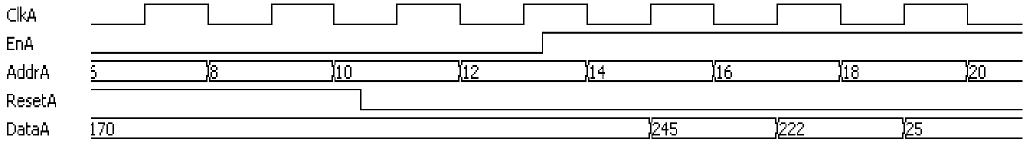
In the single-port (Port A) configuration, signals are synchronized to ClkA; ResetA can be synchronous or asynchronous depending on parameter selection. The read address (AddrA) and/or data output (DataA) can be registered to increase memory performance and improve timing. Both the read address and data output are subject to clock latency based on the ROM configuration (see *Clock Latency*, on page 260). In the dual-port configuration, all Port A signals are synchronized to ClkA, and all PortB signals are synchronized to ClkB. ResetA and ResetB can be synchronous or asynchronous depending on parameter selection, and both data outputs can be registered and are subject to the same clock latencies. Registering the data output is recommended.

Note: When the data output is unregistered, the data is immediately set to its predefined reset value concurrent with an active reset signal.

Single-Port Read Operation

For single-port ROM, it is only necessary to configure Port A (see *Specifying ROMs with SYNCore*, on page 520 in the *User Guide*). 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.

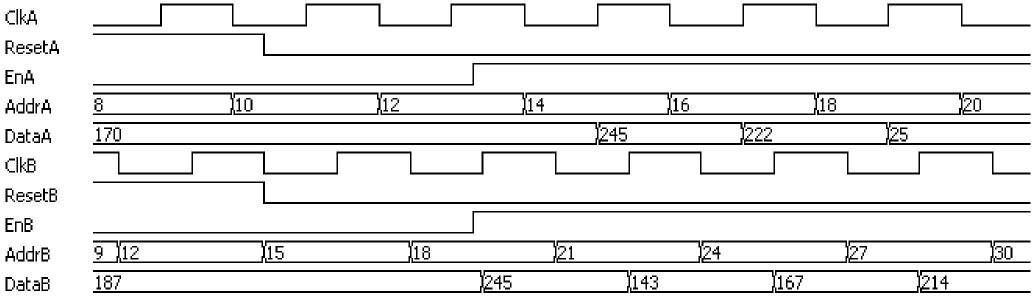


When reset is active, the output data holds the initialization value (i.e., 255). When reset goes inactive (and enable is active), data is read form the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset are inactive, the output holds its previous read value.

Note: In the above timing diagram, reset is synchronous. Clock latency varies according to the implementation and parameters as described in *Clock Latency*, on page 260.

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 520 in the *User Guide*). The following diagram shows the read timing for a dual-port ROM.



When either reset is active, the corresponding output data holds the initialization value (i.e., 255). When a reset goes inactive (and its enable is active), data is read form the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset are inactive, the output holds its previous read value.

Note: In the above timing diagram, reset is synchronous. Clock latency varies according to the implementation and parameters as described in *Clock Latency*, on page 260.

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	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	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

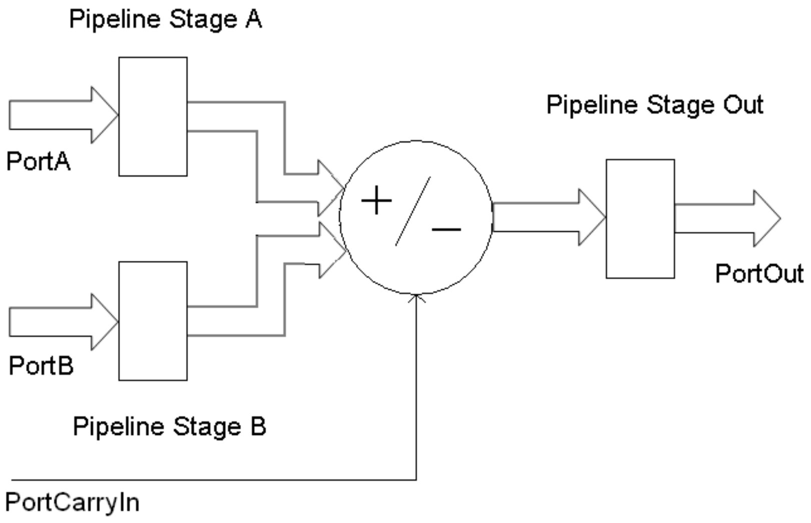
The SYNCore adder/subtractor compiler generates Verilog code for a parametrizable, pipelined adder/subtractor. This section describes the functionality of this block in detail.

Functional Description

The adder/subtractor has a single clock that controls the entire pipeline stages (if used) of the adder/subtractor.

As its name implies, this block just adds/subtracts the inputs and provides the output result. One of the inputs can be configured as a constant. The data inputs and outputs of the adder/subtractor can be pipelined; the pipeline stages can be 0 or 1, and can be configured individually. The individual pipeline stage registers include their own reset and enable ports.

The reset to all of the pipeline registers can be configured either as synchronous or asynchronous using the RESET_TYPE parameter. The reset type of the pipeline registers cannot be configured individually.



SYNCore adder/subtractor has ADD_N_SUB parameter, which can take three values ADD, SUB, or DYNAMIC. Based on this parameter value, the adder/subtractor can be configured as follows.

- Adder
- Subtractor
- Dynamic Adder and Subtractor

Adder

Based on the parameter CONSTANT_PORT, the adder can be configured in two ways.

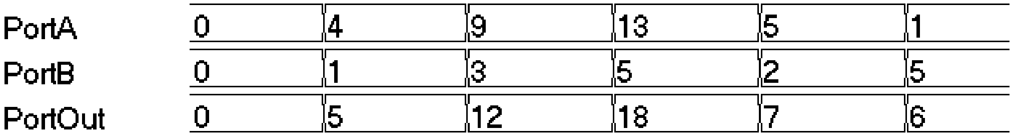
- CONSTANT_PORT='0' – adder with two input ports (port A and port B)
- CONSTANT_PORT='1' – adder with one constant port

Adder with Two Input Ports (Port A and Port B)

In this mode, port A and port B values are added. Optional pipeline stages can also be inserted at port A, port B or at both port A and port B. Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, a number of the adder configurations are given below.

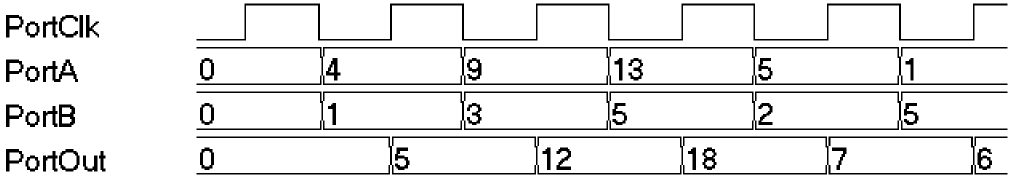
Adder with No Pipeline Stages – In this mode, the port A and port B inputs are added. The adder is purely combinational, and the output changes immediately with respect to the inputs.

Parameters: PORTA_PIPELINE_STAGE= ‘0’



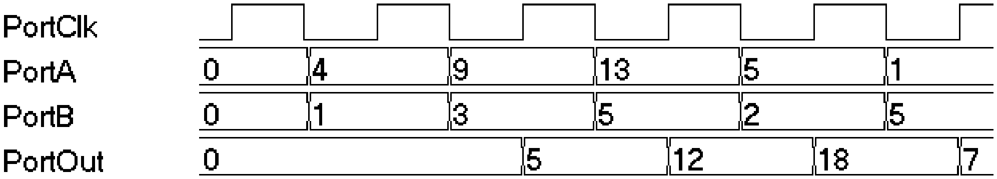
Adder with Pipeline Stages at Input Only – In this mode, the port A and port B inputs are pipelined and added. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’



Adder with Pipeline Stages at Input and Output – In this mode, the port A and port B inputs are pipelined and added, and the result is pipelined. The result is valid only on the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’



Adder with a Port Constant

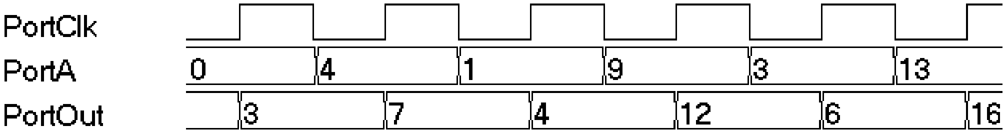
In this mode, port A is added with a constant value (the constant value can be passed though the parameter CONSTANT_VALUE). Optional pipeline stages can also be inserted at port A, Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the adder configurations are given below (here `CONSTANT_VALUE=’3’`)

Adder with No Pipeline Stages – In this mode, input port A is added with a constant value. The adder is purely combinational, and the output changes immediately with respect to the input.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’

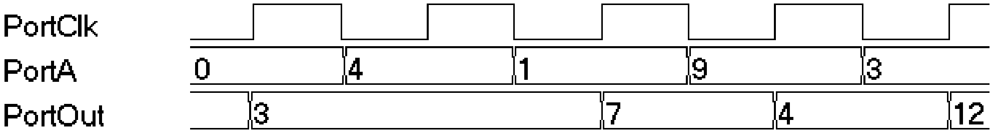
Adder with Pipeline Stage at Input Only – In this mode, input port A is pipelined and added with a constant value. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’



Adder with Pipeline Stages at Input and Output – In this mode, input port A is pipelined and added with a constant value, and the result is pipelined. The result is valid only on the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’



Subtractor

Based on the parameter CONSTANT_PORT, the subtractor can be configure in two ways.

CONSTANT_PORT='0' – subtractor with two input ports (port A and port B)

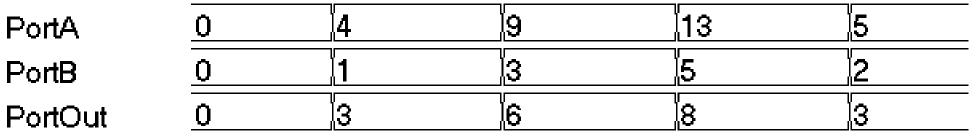
CONSTANT_PORT='1' – subtractor with one constant port

Subtractor with Two Input Ports (Port A and Port B)

In this mode, port B is subtracted from port A. Optional pipeline stages can also be inserted at port A, port B, or both ports. Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the subtractor configurations are given below.

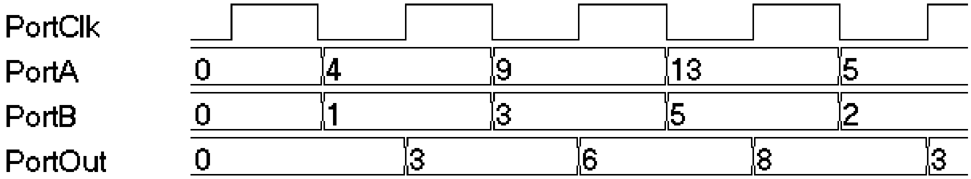
Subtractor with No Pipeline Stages – In this mode, input port B is subtracted from port A, and the subtractor is purely combinational. The output changes immediately with respect to the inputs.

Parameters: PORTA_PIPELINE_STAGE= '0'
PORTB_PIPELINE_STAGE= '0'
PORTOUT_PIPELINE_STAGE= '0'



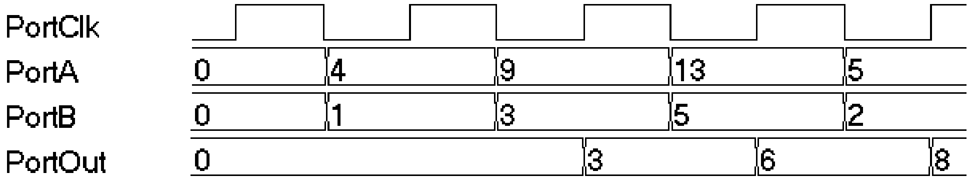
Subtractor with Pipeline Stages at Input Only – In this mode, input port B and input PortA are pipelined and then subtracted. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTB_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '0'



Subtractor with Pipeline Stages at Input and Output – In this mode, input PortA and PortB are pipelined and then subtracted, and the result is pipelined. The result is valid only at the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’

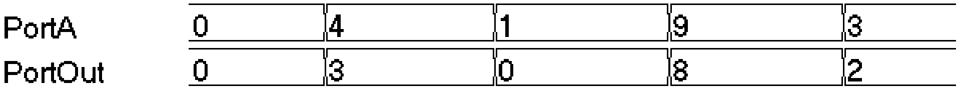


Subtractor with a Port Constant

In this mode, a constant value is subtracted from port A (the constant value can be passed though the parameter CONSTANT_VALUE). Optional pipeline stages can also be inserted at port A, Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, a number of the subtractor configurations are given below (here `CONSTANT_VALUE='1'`).

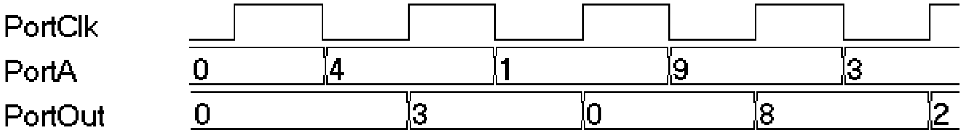
Subtractor with No Pipeline Stages – In this mode, a constant value is subtracted from port A. The subtractor is purely combinational, and the output changes immediately with respect to the input.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’



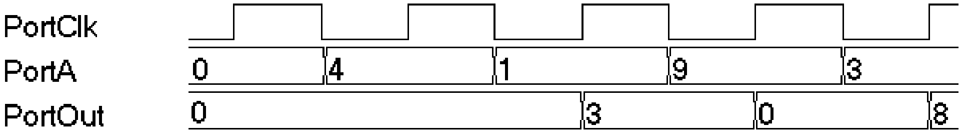
Subtractor with Pipeline Stages at Input Only – In this mode, a constant value is subtracted from pipelined input port A. Because there is no pipeline stage at the output, the output is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’



Subtractor with Pipeline Stages at Input and Output – In this mode, a constant value is subtracted from pipelined port A, and the output is pipelined. The result is valid only at the second rising edge of the clock.

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

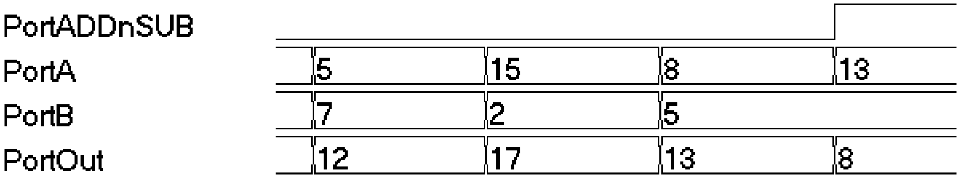
CONSTANT_PORT='1' – dynamic adder/subtractor with one constant port

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.

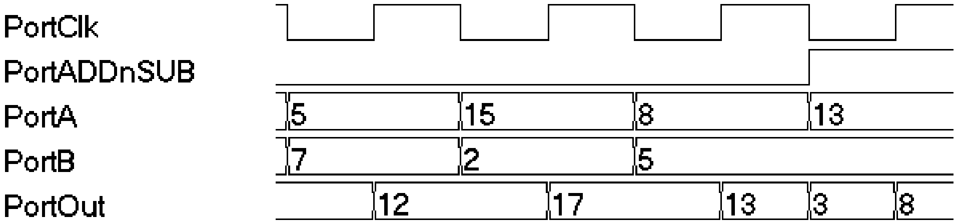
Dynamic Adder/Subtractor with No Pipeline Registers – In this mode, the dynamic adder/subtractor is a purely combinational, and output changes immediately with respect to the inputs.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTB_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’



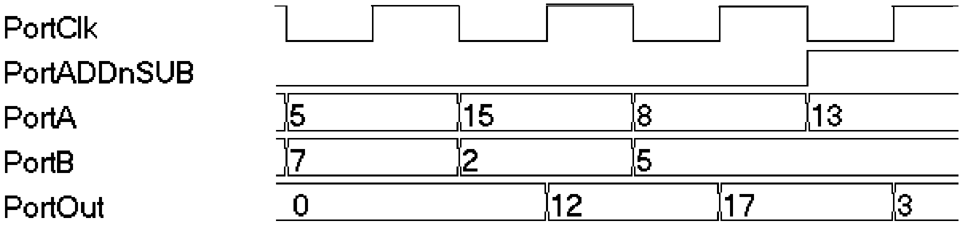
Dynamic Adder/Subtractor with Pipeline Stages at Input Only – In this mode, input port A and port B are pipelined and then added/subtracted based on the value of port PortADDnSUB. Because there is no pipeline stage at the output port, the result immediately changes with respect to the PortADDnSUB signal.

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: PORTA_PIPELINE_STAGE= '1'
PORTB_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '1'

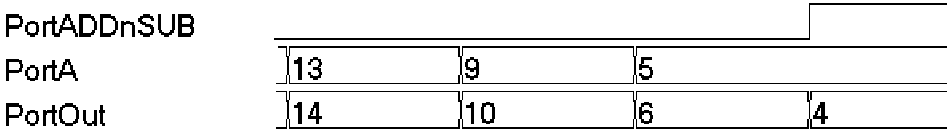


Dynamic Adder/Subtractor with a Port Constant

In this mode, a constant value is either added or subtracted from port A based on input port value PortADDnSUB (the constant value can be passed though the parameter CONSTANT_VALUE). Optional pipeline stages can also be inserted at port A, Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the dynamic adder/subtractor configurations are given below (here `CONSTANT_VALUE='1'`).

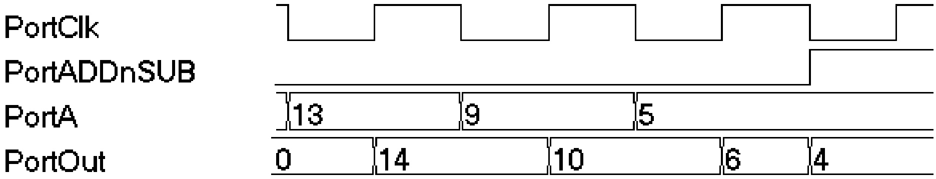
Dynamic Adder/Subtractor with No Pipeline Registers – In this mode, dynamic adder/subtractor is a purely combinational, and the output change immediately with respect to the input.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’



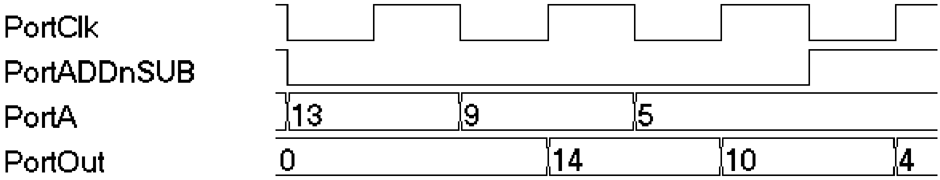
Dynamic Adder/Subtractor with Pipeline Stages at Input Only – In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because there is no pipeline stage on the output port, the result changes immediately with respect to the PortADDnSUB signal.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’



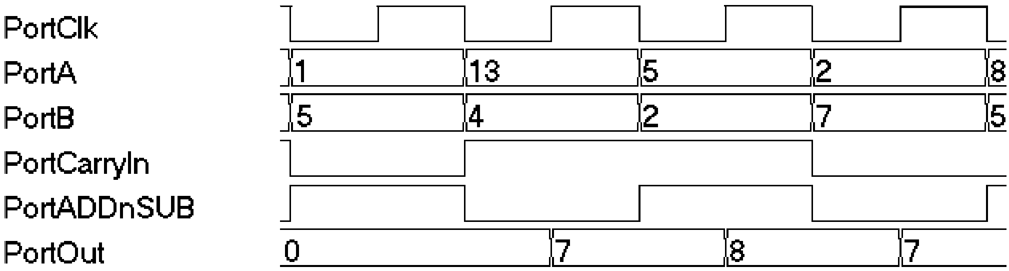
Dynamic Adder/Subtractor with Pipeline Stages at Input and Output – In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because the output port is pipelined, the result is valid only on the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’



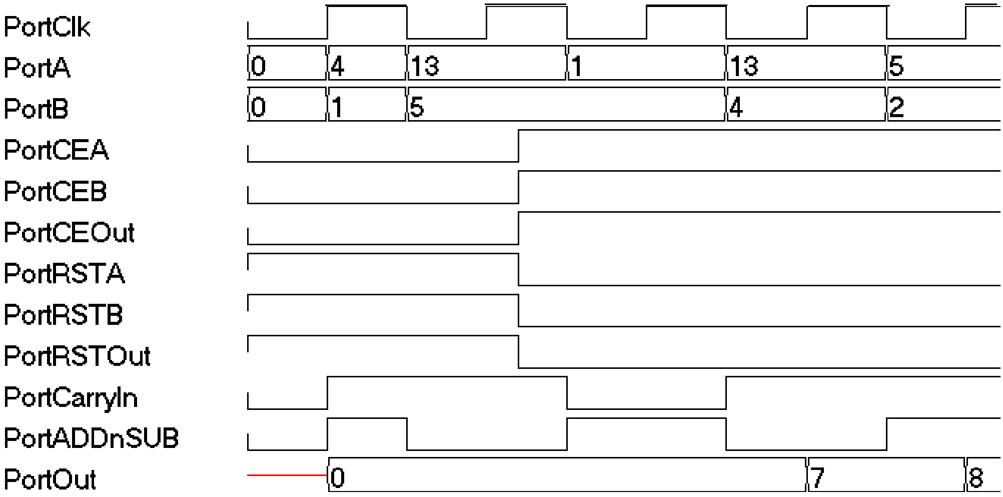
Dynamic Adder/Subtractor with Carry Input

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 274
- UP Counter Operation, on page 275
- Down Counter Operation, on page 275
- Dynamic Counter Operation, on page 276

For further information, refer to the following:

- *Specifying Counters with SYNCore,* on page 532 of the *User Guide*, for information about using the wizard to generate a counter core.
- *Launch SYNCore Command,* on page 401 and *SYNCore Counter Wizard,* on page 427 for descriptions of the interface and generating the core.

Functional Overview

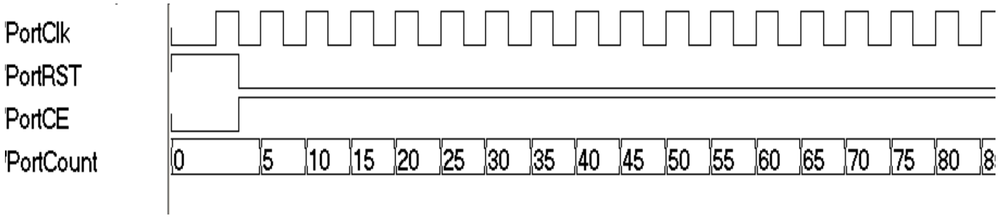
The SYNCore counter component supports up, down, and dynamic (up/down) counter implementations using DSP blocks or logic elements. For each configuration, design optimizations are made for optimum use of core resources.

As its name implies, the COUNTER block counts up (increments) or down (decrements) by a step value and provides an output result. You can load a constant or a variable as an intermediate value or base for the counter. Reset to the counter on the PortRST input is active high and can be can be configured either as synchronous or asynchronous using the RESET_TYPE parameter. Count enable on the PortCE input must be value high to enable the counter to increment or decrement.

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'

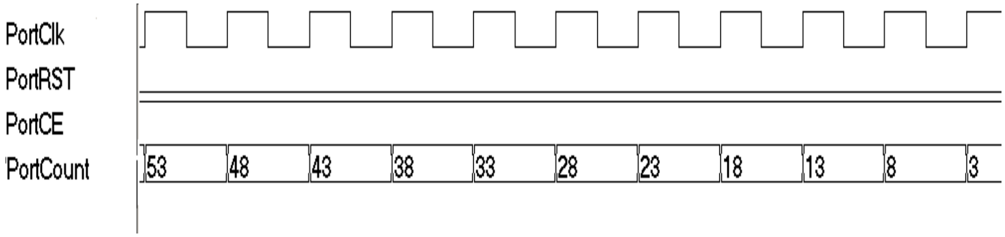


Note: Counter core can be configured to use a constant or dynamic load value in Up Counter mode (for the counter to load the PortLoadValue, PortCE must be active). This functionality is explained in *Dynamic Counter Operation,* on page 276.

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 276.

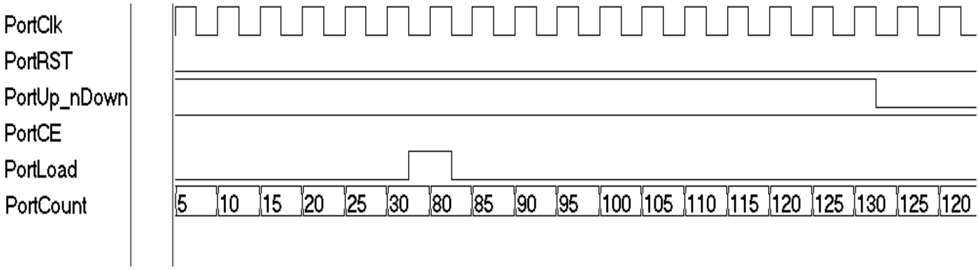
Dynamic Counter Operation

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’



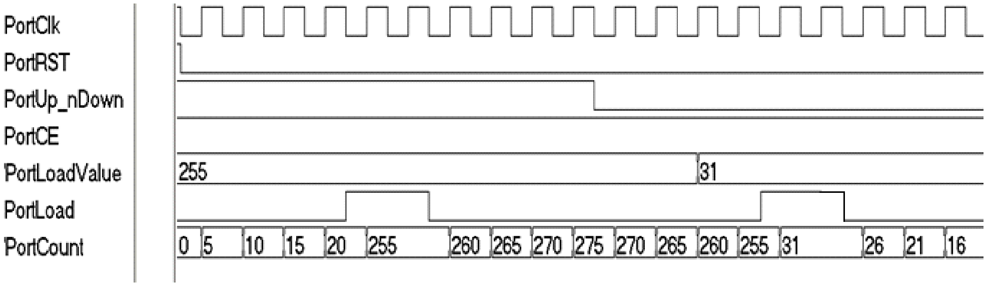
Note: *For counter to load the PortLoadValue, PortCE must be active.

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

Scripts

This chapter describes Tcl scripts.

- synhooks File Syntax, on page 280
- Tcl Script Examples, on page 282

synhooks File Syntax

The Tcl hooks commands provide an advanced user with callbacks to customize a design flow or integrate with other products. To enable these callbacks, set the environment variable SYN_TCL_HOOKS to the location of the Tcl hooks file(synhooks.tcl), then customize this file to get the desired customization behavior. For more information on creating scripts using synhooks.tcl, see *Automating Flows with synhooks.tcl*, on page 579.

Tcl Callback Syntax	Function
proc syn_on_set_project_template {*projectPath***} {***yourDefaultProjectSettings***}**	Called when creating a new project. *projectPath* is the path name to the project being created.
proc syn_on_new_project {*projectPath***} {***yourCode***}**	Called when creating a new project. *projectPath* is the path name to the project being created.
proc syn_on_open_project {*projectPath***} {***yourCode***}**	Called when opening a project. *projectPath* is the path name to the project being created.
proc syn_on_close_project {*projectPath***} {***yourCode***}**	Called after closing a project. *projectPath* is the path name to the project being created.
proc syn_on_start_application {*applicationName version currentDirectory***} {***yourCode***}**	Called when starting the application. • *applicationName* is the name of the software. For example synplify_pro. • *version* is the name of the version of the software. For example 8.4 • *currentDirectory* is the name of the software installation directory. For example C:\synplify_pro\bin\synplify_pro.exe.
proc syn_on_exit_application {*applicationName version***} {***yourCode***}**	Called when exiting the application. • *applicationName* is the name of the software. For example synplify_pro. • *version* is the name of the version of the software. For example 8.4.

Tcl Callback Syntax	Function
proc syn_on_start_run {*runName projectPath implementationName***} {***yourCode***}**	Called when starting a run. • *runName* is the name of the run. For example compile or synthesis. • *projectPath* is the location of the project. • *implementationName* is the name of the project implementation. For example, rev_1.
proc syn_on_end_run {*runName projectPath implementationName***} {***yourCode***}**	Called at the end of a run. • *runName* is the name of the run. For example, compile or synthesis. • *projectPath* is the location of the project. • *implementationName* is the name of the project implementation. For example, rev_1.
proc syn_on_press_ctrl_F8 {} {*yourCode***}**	Called when Ctrl-F8 is pressed. See Tcl Hook Command Example below.
proc syn_on_press_ctrl_F9 {} {*yourCode***}**	Called when Ctrl-F9 is pressed.
proc syn_on_press_ctrl_F8 {} {*yourCode*	Called when Ctrl-F11 is pressed.

Tcl Hook Command Example

Create a modifier key (ctrl-F8) to get all the selected files from a project browser and project directory.

```
set sel_files [get_selected_files]
while {[expr [llength $sel_files] > 0]} {
   set file_name [lindex $sel_files 0]
      puts $file_name
   set sel_files [lrange $sel_files 1 end]
}
```

Tcl Script Examples

This section provides the following examples of Tcl scripts:

- Using Target Technologies, on page 282
- Different Clock Frequency Goals, on page 282
- Setting Options and Timing Constraints, on page 283

Using Target Technologies

```
# Run synthesis multiple times without exiting, while trying different
# target technologies. View the implementations in the HDL Analyst tool.

# Open a new project
   project -new

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

# Add a Verilog file to the source file list.
   add_file -verilog "D:/test/simpletest/prep2_2.v"

# Create a new Tcl variable, called $try_these, used to synthesize
# the design using different target technologies.

   set try_these {

         ECP2 ECP3 # list of technologies
   }

# Loop through synthesis for each target technology.
   foreach technology $try_these {
            impl -add
            set_option -technology $technology
            project -run -fg
            open_file -rtl_view
   }
```

Different Clock Frequency Goals

```
# Run synthesis six times on the same design using different clock
# frequency goals. Check to see what the speed/area tradeoffs are for
# the different timing goals.
# Load an existing Project. This Project was created from an
# interactive session by saving the Project file, after adding all the
# necessary files and setting options in the Project -> Options for
# implementation dialog box.


   project -load "design.prj"

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

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

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

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

# Run synthesis.
   project -run

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

Setting Options and Timing Constraints

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

# Open a new Project
   project -new
```

```
# Set the target technology, part number, package, and speed grade options.
   set_option -technology ECP3
   set_option -part LFE3_17EA
   set_option -package MG328C
   set_option -speed_grade -6

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

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

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

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

# Set the design frequency.
   set_option -frequency 30.0

# Save the existing Project to a file. The default synthesis Result File
# is named "<project_name>.<ext>". To name the synthesis Result File
# something other than "design.xnf", use project -result_file "<name>.xnf"
   project -save "design.prj"

# Synthesize the existing Project
   project -run

# Open an RTL View
   open_file -rtl_view

# Open a Technology View
   open_file -technology_view

# ----------------------------------------------------
# This constraint file, "design.fdc," is read by "test3.tcl"
# with the add_file -constraint "design.fdc" command. Constraint files
# are for timing constraints and synthesis attributes.
# ----------------------------------------------------
# Timing Constraints:
# ----------------------------------------------------
# The default design frequency goal is 30.0 MHz for four clocks. Except
# that clk_fast needs to run at 66.0 MHz. Override the 30.0 MHz default
# for clk_fast.
   create_clock {clk_fast} -freq 66.0

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

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

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

APPENDIX A

Designing with Lattice

This chapter discusses the following topics for synthesizing Lattice designs:

- Lattice Technology Support, on page 288
- IP Encryption Flow for Lattice, on page 289
- Lattice Features, on page 290
- Lattice Constraints, Attributes, and Options, on page 324
- Device Mapping Options, on page 327
- Design Options, on page 331
- Implementation Results, on page 333
- LatticeECP/EC and Later Devices, on page 334
- LatticeSC/SCM and LatticeXP2/XP Devices, on page 338
- Lattice MachXO and Platform Devices, on page 341
- Lattice iCE40 Devices, on page 345
- set_option Command for Lattice Architectures, on page 348
- Lattice Attribute and Directive Summary, on page 352

Lattice Technology Support

This section describes general guidelines for using the synthesis tool and Lattice place-and-route (P&R) tool with Lattice devices. Refer to:

- Supported Device Families, on page 288
- Netlist Format, on page 289

Supported Device Families

The synthesis tool creates technology-specific netlists for the following Lattice technologies:

FPGA

- LFD2NX
- LIFMD
- iCE40UL
- iCE5LP
- ECP5U/ECP5UM
- iCE40LM/iCE40
- LatticeECP3
- LatticeECP2S/ECP2MS/ECP2M/ECP2
- LatticeECP
- LatticeEC
- LatticeSCM
- LatticeSC
- LatticeXP2/LatticeXP
- MachXO3L
- MachXO3LF
- MachXO2/MachXO
- Platform Manager/Platform Manager 2

New devices are added on an ongoing basis. For the most current list of supported devices, select Project->Implementation Options and check the list of technologies on the Device tab.

Netlist Format

The synthesis tool outputs EDIF netlist files for use with the Lattice ispLever, Diamond, or iCEcube2 P&R application. These files have .edn and .edf extensions.

IP Encryption Flow for Lattice

Lattice technologies fully support the Synopsys FPGA IP encryption flow. This IP encryption flow is a design flow that encourages interoperability while protecting IP implementations using encryption/decryption technology licensed from RSA Securities. This flow offers the following advantages: interoperability, protection of IP, reuse of IP, and a standard flow for IP encryption.

The Lattice IP encryption scheme uses the flow described in *Working with Lattice IP,* on page 564. Use this procedure for details on how to incorporate Lattice encrypted IP in your design.

Lattice Features

This section describes some supported Lattice features:

- RAM Support, on page 290
- DSP Support, on page 291
- Inference Support for SB_MAC16 Blocks, on page 292
- Lattice Oscillator Timing Propagation Support, on page 297
- Block RAM Support, on page 298
- SYNCORE RAMs, on page 309
- Macros and Black Boxes, on page 309
- Programmable I/O Cell Latches, on page 309
- Initial Value Support for Registers, on page 310
- Hierarchical Attribute Support for Module Generation, on page 311
- Selective Clock Enable Inference, on page 311
- Shift Chain Inference, on page 312
- Lattice iCE40 and iCE40LM Components, on page 313
- Forward Annotation, on page 322
- Supported Timing Constraints, on page 323
- Synthesis Reports, on page 323

DSP Support

The tool supports the inference of the following for the JE5D00 device:

- MULT9X9, MULT18X18, MULT18X36 and MULT36X36 DSP primitives
- MULT9X9, MULT18X18, MULT18X36 and MULT36X36 DSP primitives for Mult operations
- MULTADDSUB18X18, MULTADDSUB18X36 and MULTADDSUB36X36 DSP primitives for Mult-Add operations
- MULTADDSUB18X18, MULTADDSUB18X36 and MULTADDSUB36X36 DSP primitives for Mult-Accumulator and Signed Mult-Sub operations
- MULTADDSUB9X9WIDE and MULTADDSUB18X18WIDE DSP primitives for Mult-Add, Mult-Accumulator and Signed Mult-Sub operations

The tool also supports the inference of MULTPREADDX18 and MULTPREADDX36 primitives for Preadder-Mult operations.

The tool supports merging of multiple SIGNED ports to one SIGNED port.

The name of the primitive MULTPREADD18 is changed to MULTPREADD9X9, and that of MULTPREADD36 is changed to MULTPREADD18X18.

Inference Support for SB_MAC16 Blocks

Lattice iCE5LP family

The Synplify Pro tool supports the inference of SB_MAC16 primitive during logic synthesis for better performance and resource utilization.

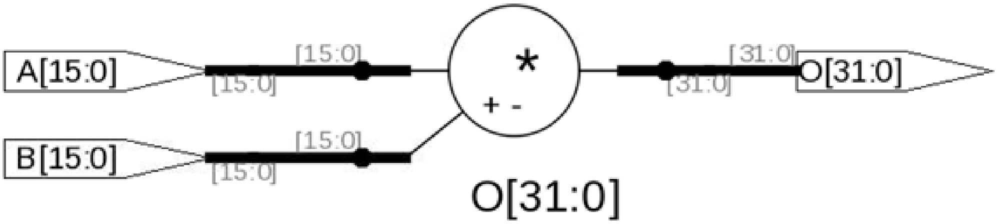
The following structures in RTL are mapped to SB_MAC16 block:

- Multipliers

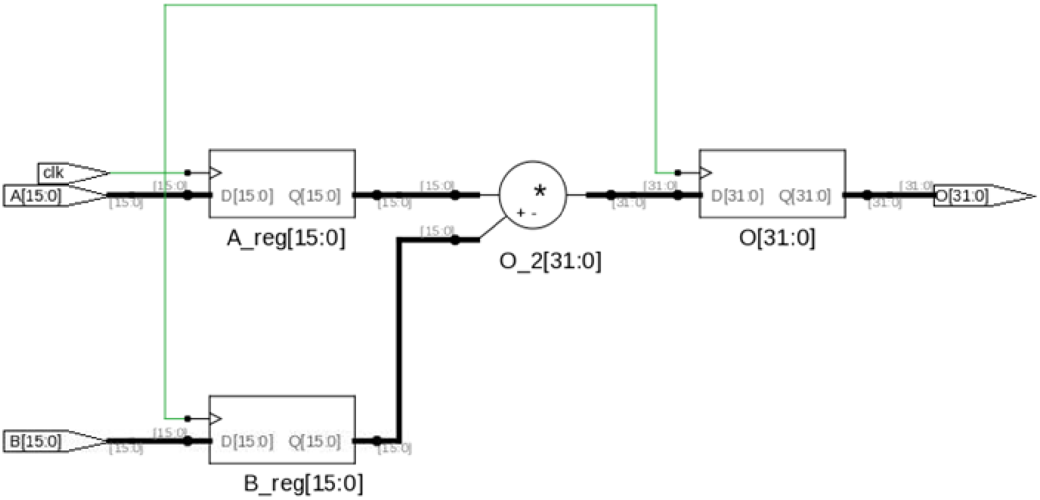
  Signed or unsigned multipliers with or without input and output registers are packed into the SB_MAC16 block.

  The following multiplier structures are packed into the SB_MAC16 block:

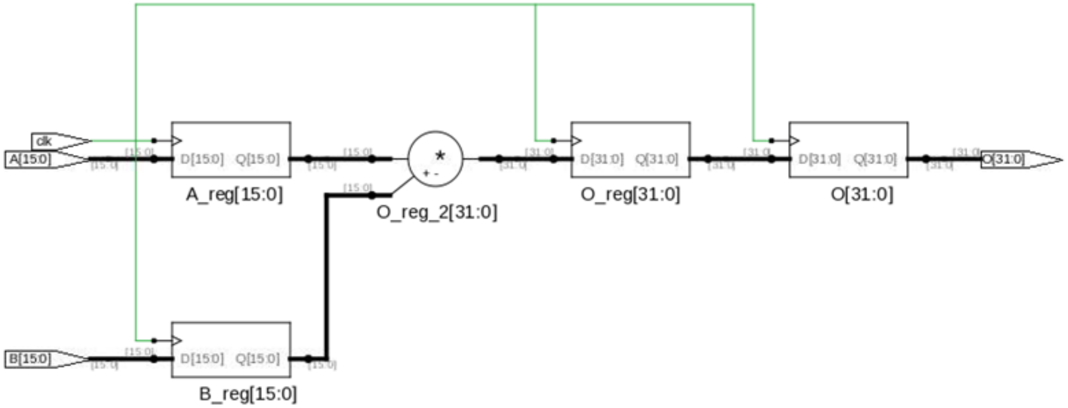
  – Signed/unsigned multiplier in bypassed mode



- Signed/unsigned multiplier in intermediate register bypassed mode



- Signed/unsigned multiplier in all register pipeline mode



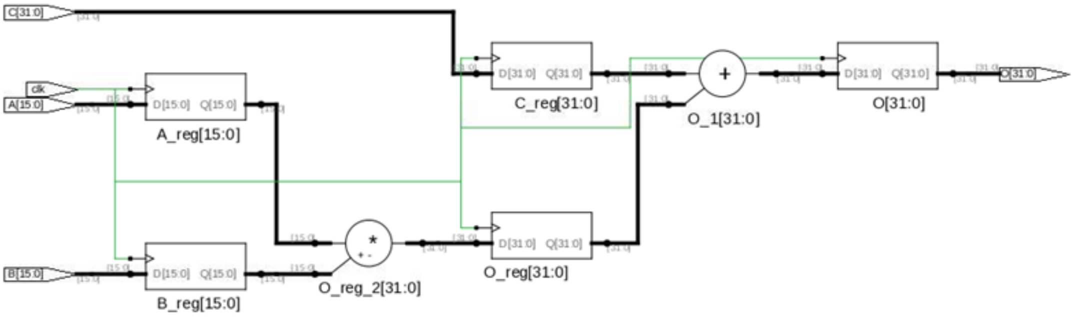
Note:

- For any other structures, only the multiplier is packed in bypass mode.
- Multipliers, having any of the input widths more than 16, are fractured into smaller size multipliers. The Synplify Pro tool packs partial products in the SB_MAC16 block.
- Multiply 8x8 mode is not supported in synthesis due to hardware limitations. So, one 8x8 multiplier consumes one full SB_MAC16 block.

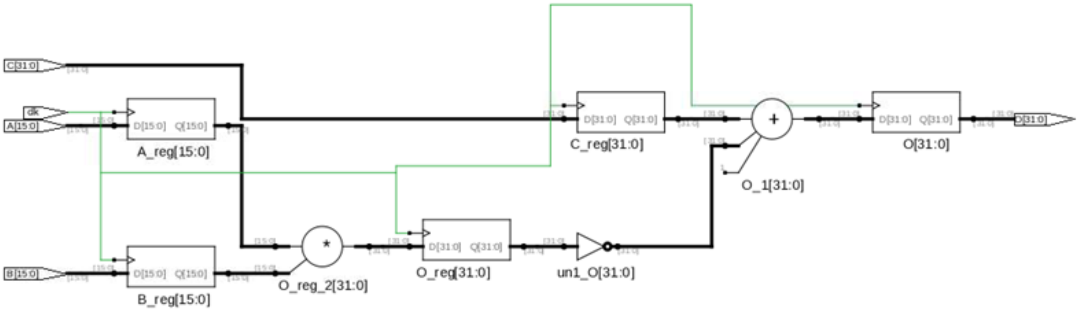
- Mult-add/mult-sub (multiplier followed by an adder or subtractor)

  Signed or unsigned multiplier-adder/subtractor is packed into the SB_MAC16 block. The following mult-add/mult-sub structures are packed into the SB_MAC16 block:

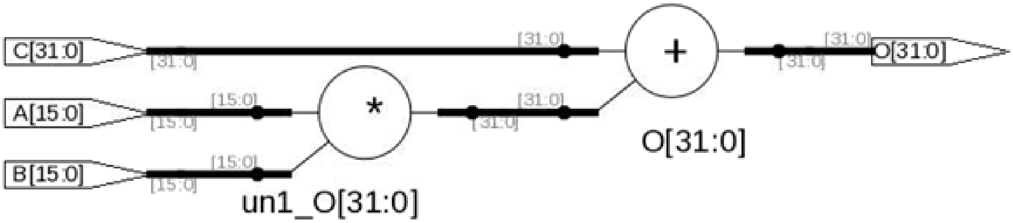
  - Signed or unsigned multiplier-adder in intermediate register bypassed mode



- Signed or unsigned multiplier-subtractor in intermediate register bypassed mode



- Unsigned multiplier-adder/subtractor in bypassed mode



- Overflow bit logic support

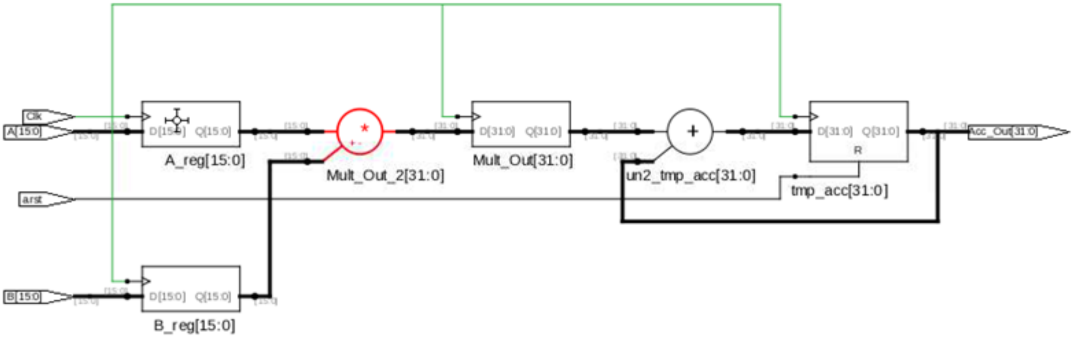
  If the output width of the mult-add/sub is 33, the Synplify Pro tool creates glue logic for the overflow bit (bit number 33). If the width is greater than 33, the adder is mapped to logic.

Note: Due to P&R limitations, signed mult-add/sub in bypass mode is not packed into SB_MAC16. Only the multiplier part of it is packed in bypass mode.

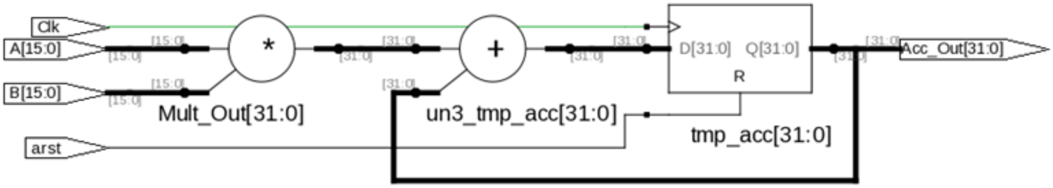
- MACC (Multiply Accumulate)

Signed or unsigned multiplier-accumulators are packed into the SB_MAC16 block. The following MACC structures are packed into the SB_MAC16 block:

  - Signed or unsigned MAC_32 in intermediate register bypassed mode



  - Unsigned MAC_32 in bypassed mode



Note:

- If the output width of the MACC is 33 or greater, the adder is mapped to logic.
- Due to P&R limitations, signed MACC in bypass mode is not packed into SB_MAC16. Only the multiplier part of it is packed in bypass mode.

Lattice Oscillator Timing Propagation Support

Timing propagation support is enhanced for OSCD, OSCF, OSCG, OSCE, and OSCH Lattice Oscillator primitives, as given below:

- The mapper computes frequency based on the parameter value on the instantiated oscillator primitive and uses it for timing the paths.
- The tool forward-annotates the calculated frequency to the LPF file.

  If there is a user constraint on the oscillator output, then that constraint is honored and the same is forward-annotated to the LPF file.

  If there is no user constraint 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 tool writes the parameter value as a property in the output EDIF netlist.
- If you specify an invalid parameter value, the tool displays an error.

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 tool writes the parameter value (*NOM_FREQ*) in the EDIF file as a property. For the RTL examples above, the property written in the *edif* output is as follows:

EDIF output

```
(instance inst0 (viewRef verilog (cellRef OSCD))
(property NOM_FREQ (string "2.5"))
```

Block RAM Support

The LatticeEC/ECP device families support three Block RAM primitives:

- SP8KA – a single-port RAM with one read and one write port. The parameter WRITEMODE can be set to either NORMAL, WRITETHROUGH, or READBEFOREWRITE.
- DP8KA – a dual-port RAM with read and write ports. The parameters WRITEMODE_A and WRITEMODE_B can be set to either NORMAL, WRITETHROUGH, or READBEFOREWRITE.
- PDP8KA – a pseudo dual-port RAM with a read-only port and a write-only port. There is no memory access mode for this primitive. If the same location is being read and written to in the same cycle, the RAM behaves as WRITETHROUGH. The table shows the pin functions on the Block RAM primitive.

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

Function	SP8KA	DP8KA	PDP8KA
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

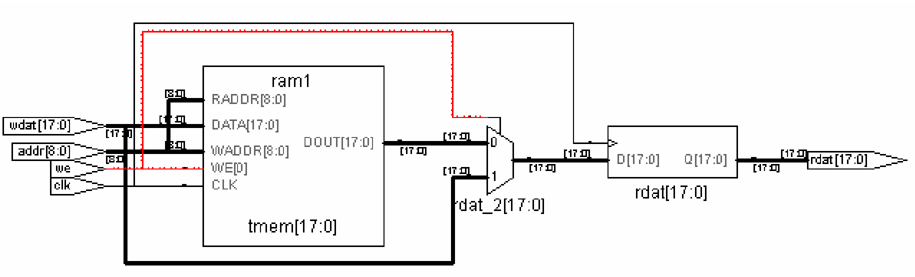
The read address and write address are the same with the exception of a flip-flop on the read data out. Based on the RTL coding style, WRITEFIRST, READFIRST, and NO_CHANGE modes apply. The mapper always uses the SP8KA Block RAM primitive.

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.

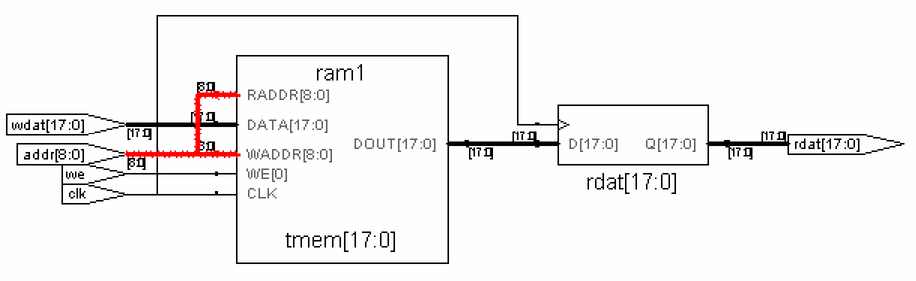
WRITE_FIRST Style

The Block RAM is set to WRITE_FIRST mode. Bypass logic is not required.



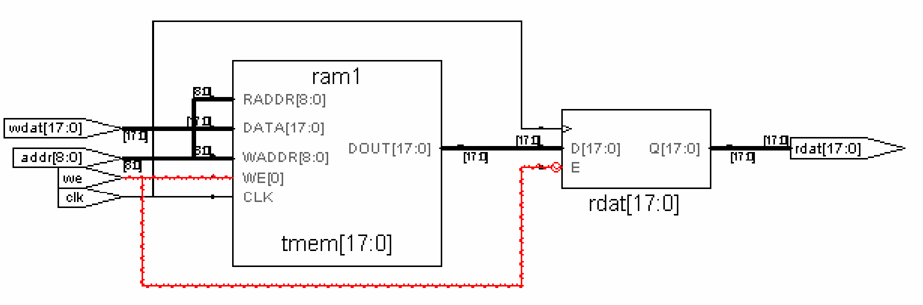
READ_FIRST Style

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.

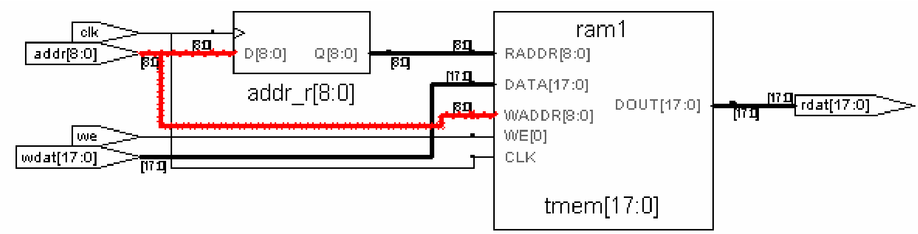


The ports are DOUT, ADDR, DIN, WE, CLK, OCLK, RST, EN

```
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.



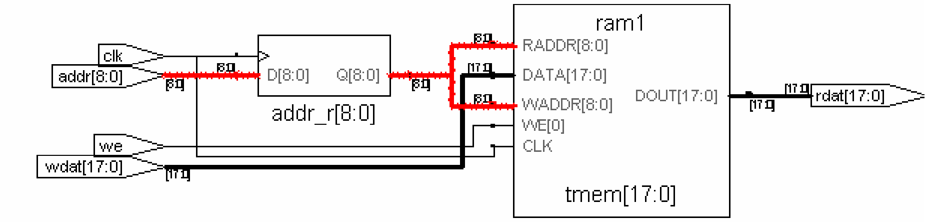
- **Synchronous Set/Reset:** If the address flip-flop has a set/reset signal, the RAM is considered as a generic case of RAM_R_W and mapped using dual-port RAMs.
- **Enable:** An address flip-flop with an enable cannot be mapped to the clock enable of the Block RAM primitive because it will affect the 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

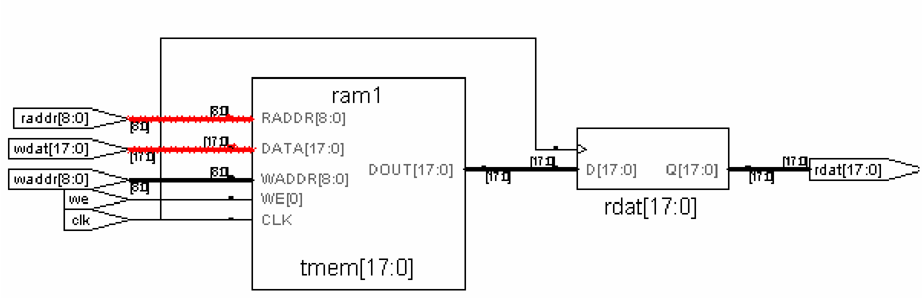
RAMN_R_W

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' },
};
```

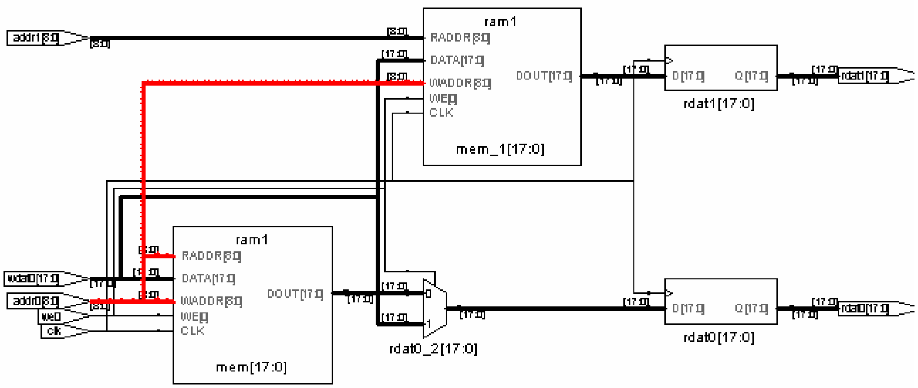
RAMN_RW_R

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.

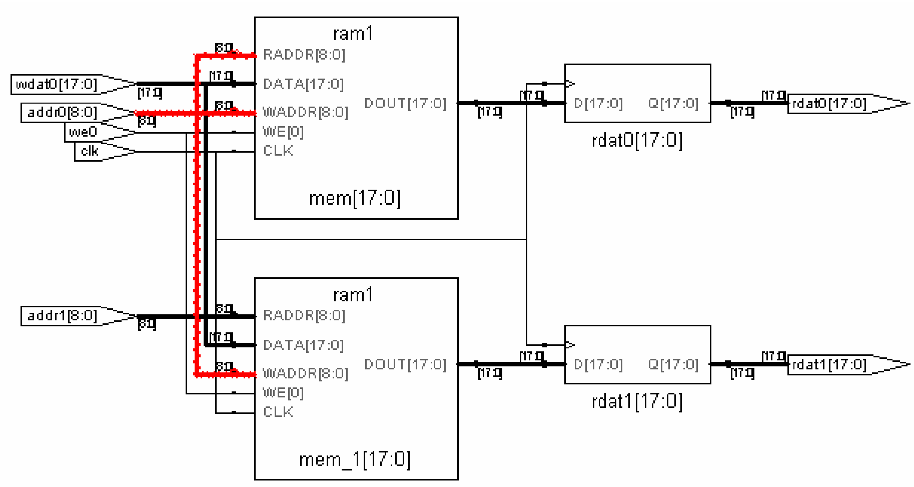
WRITE_FIRST Style

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.



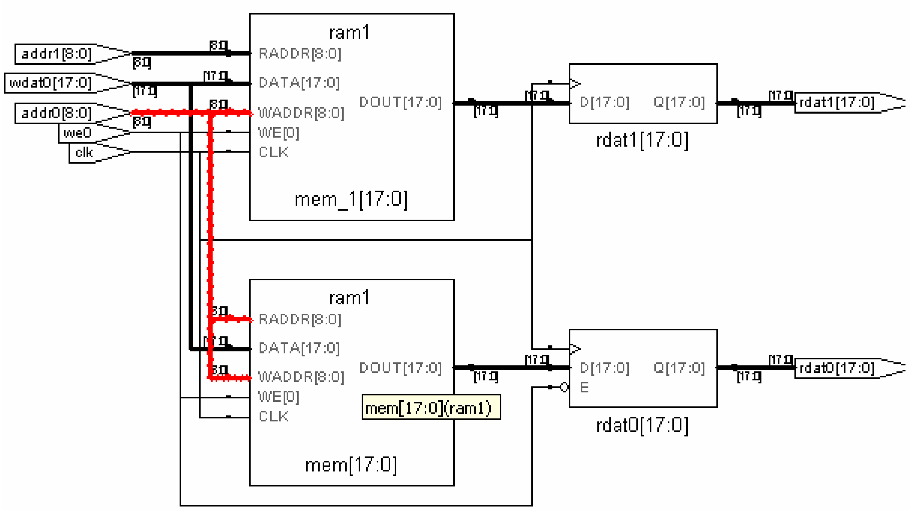
READ_FIRST Style

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.

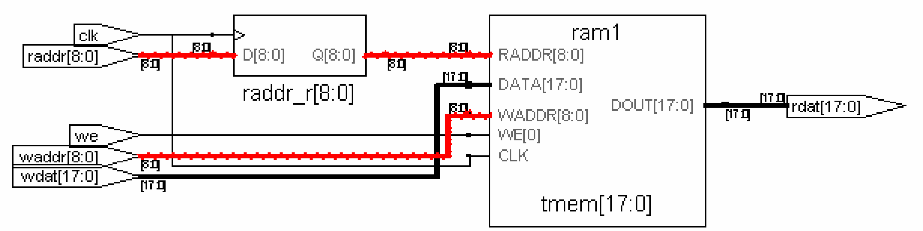


The ports are R_DOUT, RADDR, W_DOUT, DIN, WADDR, WE, CLK, R_OCLK, W_EN, EN, R_RST, W_RST

```
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.



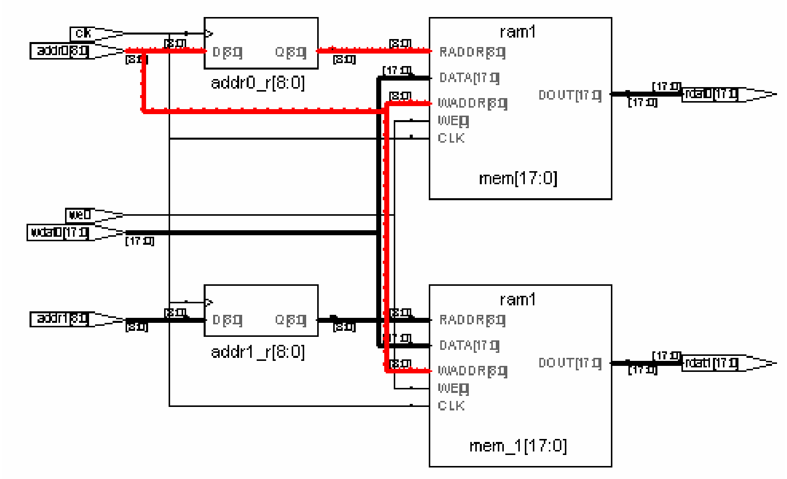
- **Synchronous Set/Reset:** is allowed on the address flip-flop. The read address to Block RAM is ANDed with reset or ORed with set. Because the write address to the Block RAM is fed from the output of the address flip-flop, the flip-flop can support set/reset.
- **Enable:** is allowed on the address flip-flop. Read address to Block RAM is modified.

The ports are DOUT, RADDR, DIN, WADDR, WE, CLK, OCLK

```
genericinfo r_w_generics[] = {
   { "family", 's' },
   { "width", 'i' },
   { "addrwidth", 'i' },
   { "depth", 'i' },
   { "raddr_reg", 'b' },
   { "din_reg", 'b' },
   { "dout_reg", 'b' },
   { "waddr_reg", 'b' },
};
```

RAM_RWP_R

There are two independent read addresses, and both are registered. The write address and one of the read addresses are pipelined. The mapper always uses the DP8KA Block RAM primitive. The RTL style is WRITEFIRST. The read/write port of the block RAM is programmed to WRITEFIRST, and does not require bypass logic. The read-only port of the Block RAM requires bypass logic.



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 506 in the *User Guide*.

Macros and Black Boxes

You can instantiate Lattice macros, such as gates, counters, flip-flops and I/Os, by using the synthesis tool-supplied Lattice macro libraries, which contain predefined Lattice black-box macros. There are Verilog and VHDL libraries. For information about using these macro libraries, see *Instantiating Lattice Macros,* on page 626 in the *User Guide*. For general information about instantiating black boxes, see *Instantiating Black Boxes in Verilog,* on page 123 and *Instantiating Black Boxes in VHDL,* on page 412.

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 628 in the *User Guide* for details and examples.

Initial Value Support for Registers

You can specify INIT values for registers, which include the following:

- Registers or latches (including PIC)

  For registers with control signals such as async set/reset, the mapper generates a warning message when the INIT value does not match the set/reset value.

- Counters and shift registers
- Registers associated with DSPs

  This may impact register packing.

  - For registers with control signals and an INIT value of either 1 or 0, the mapper ignores the initial value and register packing is not impacted.
  - For registers without control signals and an INIT value of 1, the mapper honors the initial value and register packing will not happen.

- Registers associated with RAMs

  This may impact register packing and block RAM inferencing. For Lattice EBRs, registers with INIT value of 1 are not packed into the RAM.

Synthesis software infers the following Lattice register primitives with power-up value of '1'/'0':

- FD1S3AX
  (Positive-edge Triggered D Flip-flop, GSR used for Clear with power-up value 0)
- FD1S3AY
  (Positive-edge Triggered D Flip-flop, GSR used for Preset with power-up value 1)
- FD1P3AX
  (Positive-edge Triggered D Flip-flop with Positive Level Enable, GSR used for Clear with power-up value 0)
- FD1P3AY
  (Positive-edge Triggered D Flip-flop with Positive Level Enable, GSR used for Preset with power-up value 1)

Constant propagation is affected only when the value on the data pin does not match the INIT value and retiming should not be affected.

Hierarchical Attribute Support for Module Generation

You can define attribute values at the module level which apply to elements at lower levels in the hierarchy. The Module Generator will generate PCS, PLL, DLL, and MACO type blocks, which can include single or multiple levels of hierarchy. Although modules contain multiple levels, attribute values are set at the top level.

Example

```
module PPLL (CLKIN, FB, MCLK, NCLK, LOCK, INTFB) /* synthesis
   syn_black_box */;
input CLKIN, FB;
output MCLK, NCLK, INTFB, LOCK;
parameter VCOTAP = "1";
endmodule // EXPLLA

module macro (clki, clko);
input clki;
output clko;
parameter VCOTAP = "1";
defparam u1.VCOTAP = VCOTAP;
PPLL u1 (.CLKIN(clki), .FB(fb), .MCLK(clko), .NCLK(),
   .INTFB(fb), .LOCK());
endmodule // macro

module top (clki, clko1, clko2);
input clki;
output clko1, clko2;
defparam u1.VCOTAP = "2";
defparam u2.VCOTAP = "6";
macro u1 (.clki(clki), .clko(clko1));
macro u2 (.clki(clki), .clko(clko2));
endmodule // top
```

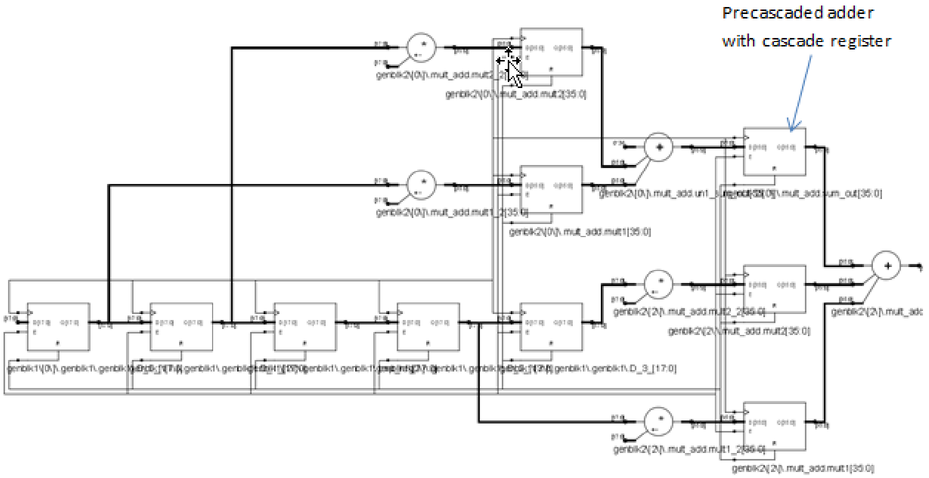
Selective Clock Enable Inference

For CPLD devices, you can extract clock enable to enhance performance using the syn_useenables attribute. See syn_useenables, on page 294 for more information.

Shift Chain Inference

Lattice ECP3 family

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.



Precascaded adder with cascade register

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.

Lattice iCE40 and iCE40LM Components

Lattice iCE40 and iCE40LM families

The following topics describe how the synthesis tool handles various Lattice components, and shows you how to work with or manipulate them during synthesis to get the results you need:

- Lattice RAM, on page 313
- Lattice ROM, on page 314
- Timing Propagation Through PLLs, on page 314
- Timing Propagation Support for Instantiated Primitives, on page 318
- Lattice Latches, on page 319
- Global Buffer Promotion for Control Signals, on page 321

Lattice RAM

Lattice iCE40 and iCE40LM families

Lattice technologies support single-port or dual-port synchronous memory.

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 414 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 *Initializing Values in RTL,* on page 314.

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 424 in the *User Guide* for step-by-step procedures. You can initialize values in Verilog and VHDL. See:

- Initializing RAMs in Verilog, on page 424
- Initializing RAMs in VHDL, on page 425

Timing Propagation Through PLLs

Lattice iCE40 and ICE40LM families

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.

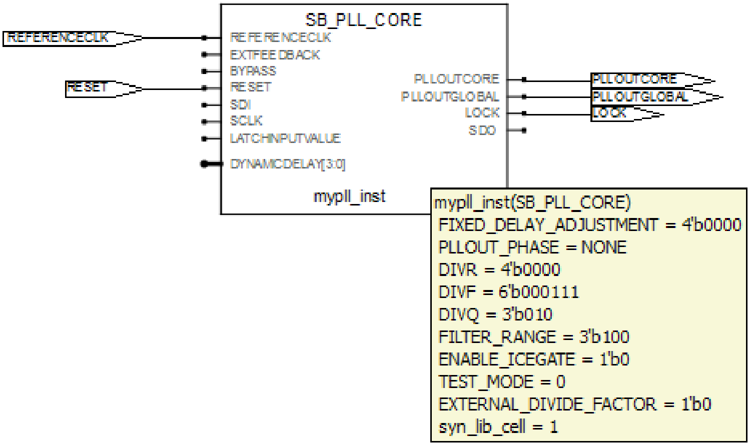
The synthesis tool assigns all the PLL outputs to the same clock group. However, only the clocks at the PLL inputs are forward-annotated to the scf file. The PLL configuration is generated from iCEcube software.

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.



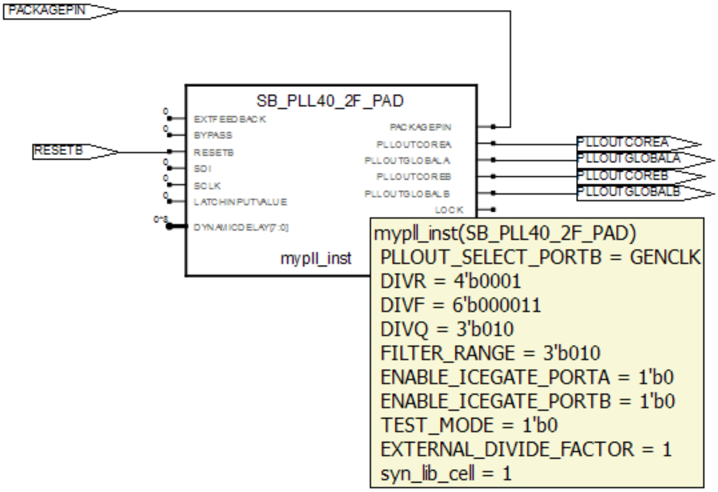
```
mypll_inst(SB_PLL_CORE)
FIXED_DELAY_ADJUSTMENT = 4'b0000
PLLOUT_PHASE = NONE
DIVR = 4'b0000
DIVF = 6'b000111
DIVQ = 3'b010
FILTER_RANGE = 3'b100
ENABLE_ICEGATE = 1'b0
TEST_MODE = 0
EXTERNAL_DIVIDE_FACTOR = 1'b0
syn_lib_cell = 1
```

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.



mypll_inst(SB_PLL40_2F_PAD)
PLLOUT_SELECT_PORTB = GENCLK
DIVR = 4'b0001
DIVF = 6'b000011
DIVQ = 3'b010
FILTER_RANGE = 3'b010
ENABLE_ICEGATE_PORTA = 1'b0
ENABLE_ICEGATE_PORTB = 1'b0
TEST_MODE = 1'b0
EXTERNAL_DIVIDE_FACTOR = 1
syn_lib_cell = 1

Timing Propagation Support for Instantiated Primitives

iCE40LM family

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(sbclki),
   .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 Latches

Lattice iCE40 and iCE40LM families

The synthesis software implements latches using LUTs for the specified technology. Types of latches supported include:

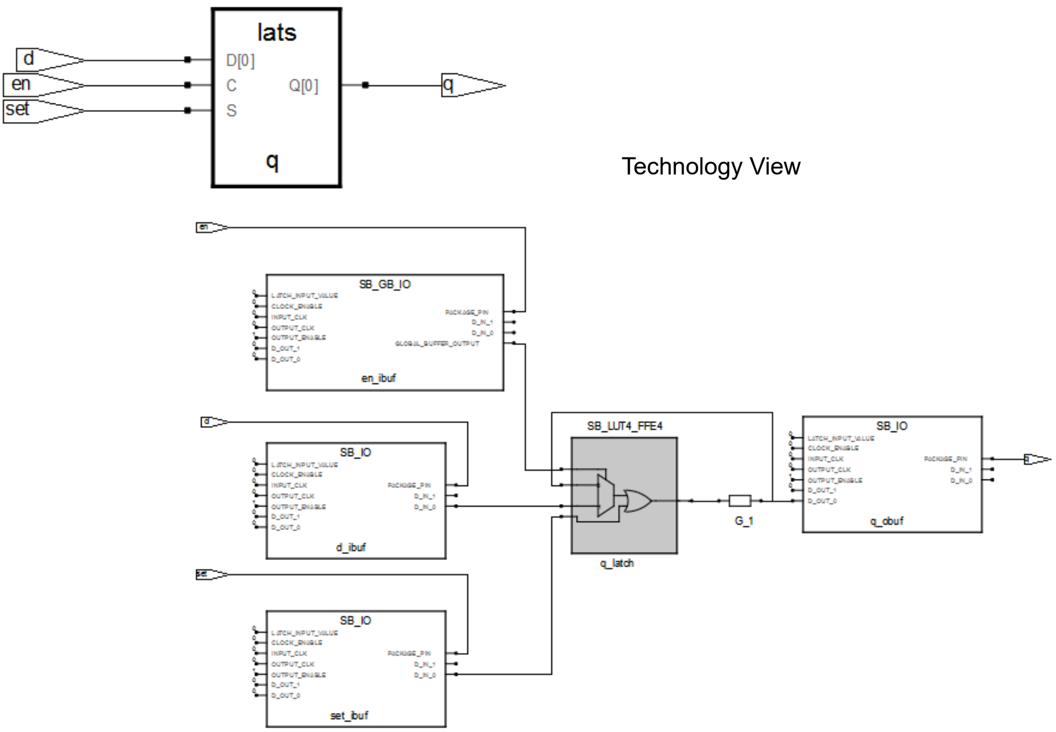
- Simple latch
- Latch with asynchronous/synchronous set
- Latch with asynchronous/synchronous reset

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.

RTL View



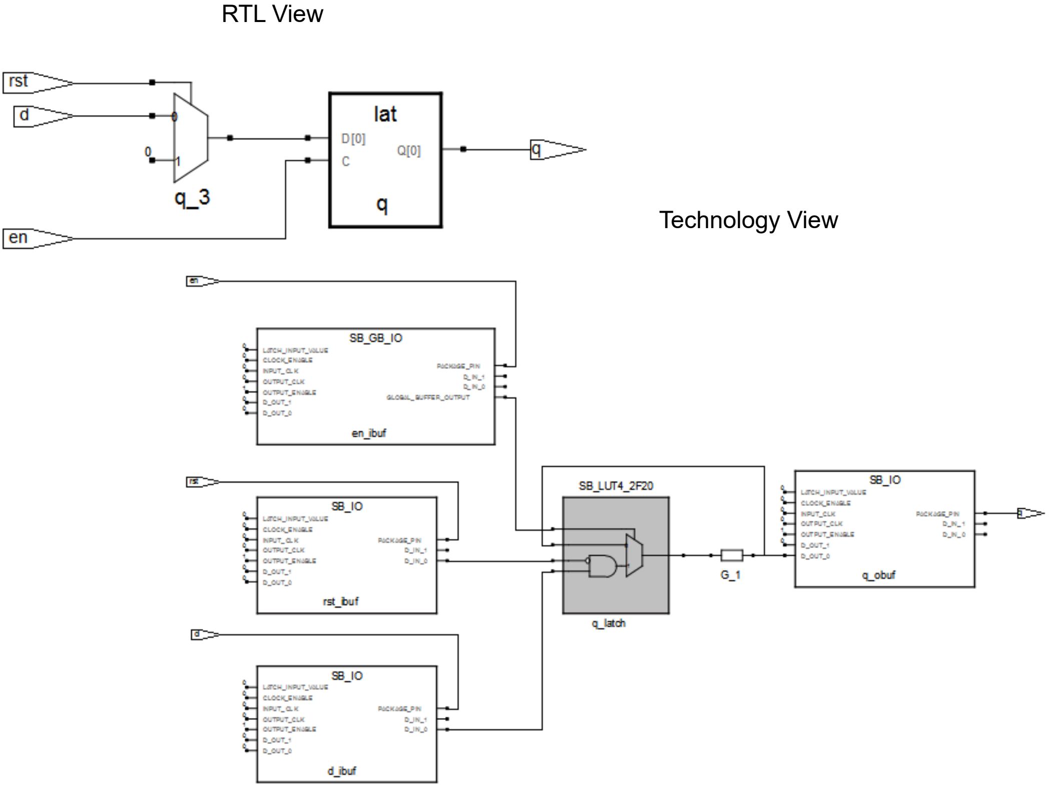
Technology View

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.



RTL View

Technology View

Global Buffer Promotion for Control Signals

Lattice iCE40 and iCE40LM families

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.

Forward Annotation

The synthesis tool generates Lattice-compliant constraint files from selected constraints that are forward-annotated (read in and then used) by the Lattice ispLever and Diamond place-and-route program. The Lattice constraint file uses the `.lpf` extension. This constraint file must be imported into the Lattice flow.

For Lattice iCE40 and iCE40LM 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.

By default, Lattice constraint files are generated from the synthesis tool constraints. You can then forward-annotate these files to the place-and-route tool. To disable this feature, deselect the Write Vendor Constraint File box (on the Implementation Results tab of the Implementation Options dialog box).

For a detailed procedure, see *Forward-Annotating Lattice Constraints, on* page 637in the *User Guide*.

Supported Timing Constraints

The constraint file generated for the Lattice ispLEVER place-and-route tool uses the $DESIGN_synplify.lpf file. Follow these steps to forward-annotate your synthesis constraint file to ispLEVER:

1. Set your constraints in the SCOPE spreadsheet (see SCOPE Tabs, on page 155).
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

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.

Timing Report

Timing reports located in the log file, are generated during synthesis for Lattice technologies. (see *Timing Reports*, on page 121). To view the log file, click View Log.

Lattice Constraints, Attributes, and Options

This section explains how to implement Lattice constraints, attributes, and options. Refer to:

- Constraints and Attributes, on page 324
- Disable I/O Insertion Globally or Port by Port, on page 325
- Device Mapping Options, on page 327

Constraints and Attributes

Specify timing constraints, general HDL attributes, and Lattice-specific attributes to improve your design. Manage these files with the SCOPE constraints and attributes editor. Lattice has vendor-specific I/O standard constraints it supports for synthesis. For a list of supported I/O standards, see *Lattice I/O Standards*, on page 324.

Lattice I/O Standards

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
SB_LVCMOS25_8	SB_LVDS_INPUT

NOTE: The I/O standards apply for L04, L08, and P04 devices. The L01 device only supports SB_LVCMOS.

Lattice I/O Standards

SB_LVCMOS25_12	SB_SUBLVDS_INPUT
SB_LVCMOS25_16	
SB_LVCMOS33_8	

NOTE: The I/O standards apply for L04, L08, and P04 devices. The L01 device only supports SB_LVCMOS.

See Also:

- Industry I/O Standards, on page 181 for a list of industry I/O standards.
- *Chapter 5, SCOPE Constraints Editor* for the timing constraints.
- *Lattice Attribute and Directive Summary, on page 352* 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 85 for more information.

I/O Buffer Support

The synthesis tool supports unconnected I/O buffers created for top-level ports. When you set syn_force_pads=1 either globally or on a port-by-port basis, the software does not optimize away the unconnected buffers. This enables the synthesis tools to do the following:

- Preserve user-instantiated pads
- Insert pads on unconnected ports

- Insert bi-directional pads on bi-directional ports instead of converting them to input ports.
- Insert output pads on unconnected outputs.

This feature is supported for Lattice ECP/EC, LatticeSC/SCM, and LatticeXP device families.

Device Mapping Options

To achieve optimal design results, set the correct implementation options. These options control the following:

- target technology and related parameters
- mapping options to use during synthesis
- output and output formats
- results directories

You can set options in the Implementation Options dialog box or by typing the set_option command and its arguments in the Tcl window. To open the Implementation Options dialog box, select Project->Implementation Options in the Project view.

One or more of the following options appear in all Lattice devices.

- Fanout Guide, on page 327
- Disable I/O Insertion, on page 327
- GSR Resource, on page 328
- Compile Point Remapping in Lattice Designs, on page 328
- Write Declared Clocks Only, on page 329

Fanout Guide

Large fanouts can cause large delays and routability problems. During technology mapping, the synthesis tool automatically maintains reasonable fanout limits, keeping the fanout under the limit you specified. For details about this option, see *Setting Fanout Limits*, on page 450 in the *User Guide*. This feature is supported for Lattice iCE40, LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager device families.

Disable I/O Insertion

The synthesis tool inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist, unless you disable I/O insertion. You can override which I/O pads are used by directly instantiating specific I/O pads. If you manually insert I/O pads, you only insert them for the pins that require them. For further details, see *Controlling I/O Insertion in Lattice Designs*, on page 636in the *User Guide*.

GSR Resource

The LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager, technology families have a resource for a GSR (global set reset) signal. The resource is a prerouted signal that goes to the reset input of each flip-flop in the FPGA. The GSR is connected to all flip-flops that are tied to the GSR line, regardless of any other defined reset signals. When the GSR is activated, all registers are reset.

The synthesis tool creates a GSR instance to access the resource, if it is appropriate for your design. Using this resource instead of general routing for a reset signal can have a significant positive impact on the routability and performance of your design.

If there is a single reset in your design, the synthesis tool connects that reset signal to a GSR instance. It forces this even if some flip-flops do not have a reset. Usually, flip-flops without set or reset can be safely initialized because the reset is only used when the device is turned on. If this is not the case, you must turn off the forced use of GSR by the synthesis tool.

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

The compile point feature is supported for Lattice iCE40, LatticeECP/EC, LatticeSC/SCM, LatticeXP2/XP, MachXO, and Platform Manager device families.

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 484 and Compile Point Basics, on page 466 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 *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used to map and optimize its parents. If the option is disabled and the *interface* of a locked compile point is changed, the immediate parent of the compile point must be changed accordingly. In this case, both are remapped, and the *updated* timing model of the child is used when remapping this parent.
Enabled	Compile point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account because the option was disabled. The timing model of a compile point is updated when either of the following is true: • The compile point is remapped, and the Update Compile Point Timing Data option is enabled. • The interface of the compile point is changed.

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.

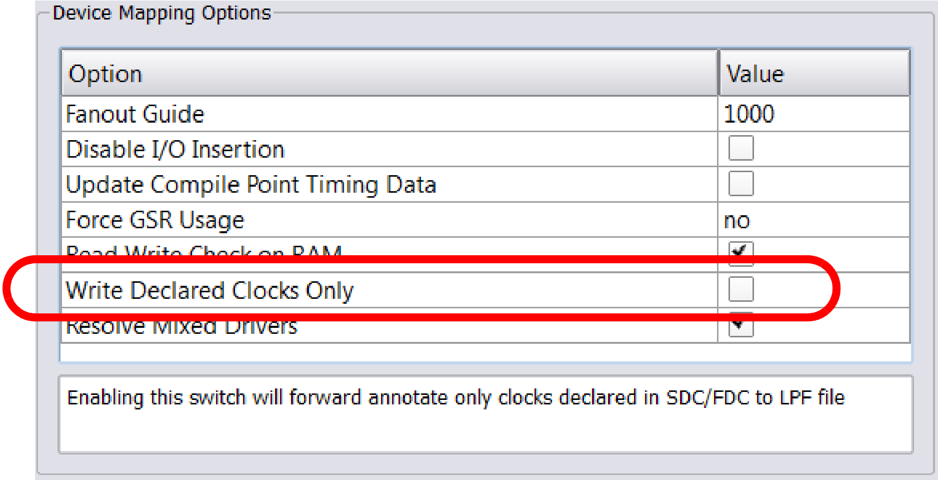
FDC File

```
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.



Device Mapping Options

Option	Value
Fanout Guide	1000
Disable I/O Insertion	□
Update Compile Point Timing Data	□
Force GSR Usage	no
Read Write Check on RAM	☑
Write Declared Clocks Only	□
Resolve Mixed Drivers	☑

Enabling this switch will forward annotate only clocks declared in SDC/FDC to LPF file

Design Options

The following options are located on the Options tab of the Implementation Options dialog box.

- FSM Compiler, on page 331
- Resource Sharing, on page 331
- Pipelining, on page 331
- Retiming, on page 332
- Gated and Generated Clocks, on page 332

FSM Compiler

Turning on the FSM Compiler enables special FSM optimizations. See *Optimizing State Machines*, on page 457in 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 163). See *Sharing Resources*, on page 454 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 185 for the syntax.

For detailed information about using pipelining, see *Pipelining*, on page 434of 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_retiming attribute to enable or disable retiming for individual flip-flops. See syn_allow_retiming, on page 36 for syntax details, or *Retiming*, on page 438 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 185 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 590 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 621 in the *User Guide*.

Implementation Results

On the Implementation Results tab of the Implementation Options dialog box, two file formats: edif and src, are available depending on your design's device family.

You can also use the project Tcl command to specify the result file format.

```
project -result_format edif
```

LatticeECP/EC and Later Devices

This section describes guidelines for using the synthesis tool with LatticeECP/EC and later technologies. The topics include:

- Device Mapping Options, on page 334
- Attributes, on page 334
- set_option Tcl Command, on page 335

Device Mapping Options

LatticeECP/EC and later families

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project->Implementation Options). You can set the following:

- Fanout Guide, on page 327
- Disable I/O Insertion, on page 327
- Retiming, on page 332
- Pipelining, on page 331
- GSR Resource, on page 328
- Compile Point Remapping in Lattice Designs, on page 328

Attributes

LatticeECP/EC and later families

See *Lattice Attribute and Directive Summary*, on page 352 for a summary of the synthesis attributes and directives you can use with the technologies.

set_option Tcl Command

LatticeECP/EC and later families

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 112.The set_option command supports the following options for the ECP5U, ECP5UM, LatticeECP3/ECP2S/ECP2MS/ECP2M/ECP2 and LatticeECP/EC devices.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion* , on page 327.
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 327.
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 328.
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.

Option	Description
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle* , on page 203.
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 328, 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 329, 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 348.

set_option Device Options

This is the syntax for setting device options for ECP5U/ECP5UM/LatticeECP3/ECP2S/ECP2MS/ECP2M/ECP2 and LatticeECP/EC devices.

set_option -technology *keyword* **-part** *partName*
    -speed_grade *value* **-package** *packageName*
        -block 1 | 0 or **-disable_io_insertion 1 | 0**
        -fanout_guide *value*
        -force_gsr yes | no | auto
        -maxfan *value*
        -resolve_multiple_driver 1|0
        -RWCheckOnRam 1|0
        -update_models_cp 1|0
        -write_declared_clocks_only 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**

LatticeSC/SCM and LatticeXP2/XP Devices

This section describes guidelines for LatticeSC/SCM and LatticeXP2/XP devices using the following options in the synthesis tool:

- Device Mapping Options, on page 338
- Attributes, on page 338
- set_option Tcl Command, on page 339

Device Mapping Options

LatticeSC/SCM and LatticeXP2/XP families

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project->Implementation Options). You can set the following:

- Fanout Guide, on page 327
- Disable I/O Insertion, on page 327
- Retiming, on page 332
- Pipelining, on page 331
- GSR Resource, on page 328
- Compile Point Remapping in Lattice Designs, on page 328

Attributes

LatticeSC/SCM and LatticeXP2/XP families

See *Lattice Attribute and Directive Summary*, on page 352 for a summary of the synthesis attributes and directives you can use with LatticeSC/SCM and LatticeXP2/XP devices.

set_option Tcl Command

LatticeSC/SCM and LatticeXP2/XP families

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 112.The set_option command supports the following options for LatticeSC/SCM and LatticeXP2/XP devices.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion* , on page 327.
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 327.
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 328.
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.

Option	Description
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle* , on page 203.
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 328, 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 329, 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 348.

set_option Device Options

This is the syntax for setting device options for LatticeSC/SCM and LatticeXP2/XP devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | 0 or **-disable_io_insertion 1 | 0**
-fanout_guide *value*
-force_gsr yes | no | auto
-maxfan *value*
-resolve_multiple_driver 1|0
-RWCheckOnRam 1|0
-update_models_cp 1|0
-write_declared_clocks_only 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
```

Lattice MachXO and Platform Devices

This section describes guidelines for MachXO, MachXO2, and Platform Manager devices using the following options in the synthesis tool:

- Device Mapping Options, on page 341
- Attributes, on page 342
- set_option Tcl Command, on page 342

Device Mapping Options

Lattice MachXO and Platform Manager families

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project -> Implementation Options). You can set the following:

- Fanout Guide, on page 327
- Disable I/O Insertion, on page 327
- Retiming, on page 332
- Pipelining, on page 331
- GSR Resource, on page 328
- Compile Point Remapping in Lattice Designs, on page 328

Attributes

Lattice MachXO and Platform Manager families

See *Lattice Attribute and Directive Summary*, on page 352 for a summary of the synthesis attributes and directives you can use with MachXO, MachXO2, and Platform Manager devices.

set_option Tcl Command

Lattice MachXO and Platform Manager families

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 112.

The set_option command supports the following options for MachXO, MachXO2, and Platform Manager devices.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion* , on page 327.
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 327.
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 328.
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.

Option	Description
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle* , on page 203.
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 328, 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 329, 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 348.

set_option Device Options

This is the syntax for setting device options for MachXO, MachXO2, and Platform Manager devices.

```
set_option -technology keyword -part partName
    -speed_grade value -package packageName
        -block 1 | 0 or -disable_io_insertion 1 | 0
        -fanout_guide value
        -force_gsr yes | no | auto
        -maxfan value
        -resolve_multiple_driver 1|0
        -RWCheckOnRam 1|0
        -update_models_cp 1|0
        -write_declared_clocks_only 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**

Lattice iCE40 Devices

This section provides guidelines for using the synthesis tool with devices for the Lattice iCE40 technologies. The topics include:

- Device Mapping Options, on page 345
- set_option Tcl Command, on page 346

Device Mapping Options

Lattice iCE40 and iCE40LM families

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 366 and *Constraints Panel*, on page 368.

The following table lists device mapping options (Device tab) for the Lattice iCE40 technology.

Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views, and use them to create collections using the Tcl Find command.
Clock Conversion	Performs gated- and generated-clock optimizations when enabled. See *Working with Gated Clocks* , on page 590 and *Optimizing Generated Clocks* , on page 621 in the *User Guide* for details.
Disable I/O Insertion	When enabled, it prevents automatic I/O insertion during synthesis. By default, it is disabled. See *Disable I/O Insertion Globally or Port by Port* , on page 325 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 450 and *Controlling Buffering and Replication* , on page 452in the User Guide.

Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle* , on page 203.
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 474, for details.

set_option Tcl Command

Lattice iCE40 and iCE40LM families

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles. The set_option Tcl command lets you specify the same implementation options as you do through the dialog box displayed in the Project view with Project->Implementation Options.

set_option Command Options

You can use the set_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 348.

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_guide value
        -maxfan value
        -resolve_multiple_driver 1|0
        -run_prop_extract 1|0
        -RWCheckOnRam 1|0
        -update_models_cp 1|0
```

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

```
set_option
        -fix_gated_and_generated_clocks 1|0
        -pipe 1 | 0
        -resource_sharing 1|0
        -retiming 1 | 0
        -symbolic_fsm_compiler 1|0 or -autosm 1|0
```

set_option Command for Lattice Architectures

The set_option Tcl command offers an alternative way to set options that are available on the Device and Options tabs on the Implementation Options dialog box. You can save options set with this command in a file and re-execute them. For a complete list of all available set_option arguments, see *set_option,* on page 112, 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 351 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	
-areadelay *percent_value*	Sets the percentage of nets you want optimized during synthesis for the open project. This option is available only if set_option -map_logic is set to 1. The default value is 0.
-block 1 | 0 **-disable_io_insertion 1 | 0**	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion* , on page 327.

OPTION	DESCRIPTION
-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 327.
-force_gsr no | yes | auto	Enables (yes) or disables (no) forced usage of the global Set/Reset routing resources for the open Project. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is no. For additional information about GSR, see *GSR Resource* , on page 328.
-fix_gated_and_ generated_clocks 1|0	Performs gated- and generated-clock optimizations when enabled. See *Working with Gated Clocks* , on page 590 and *Optimizing Generated Clocks* , on page 621 in the *User Guide* for details.
-map_logic 1 | 0	Turns on (1) or off (0) direct mapping to the macrocells cells (instead of primitive gates) during synthesis for the open project. The default value is 0, which can sometimes give a smaller result. If you set this option to true, the following three additional Tcl command options (described below) are available: -areadelay, -fanin_limit, -maxterms.
-maxfanin *value*	Sets the maximum fanin during synthesis for the open project. You can try to improve routability by reducing the fanin limit. The default value is 20. This option is available only if set_option -map_logic is set to 1.
-maxterms *value* **-max_terms_per_macrocell** *value*	Sets the maximum number of terms per macrocell during synthesis for the open project. The default value is 16. This option is available only if set_option -map_logic is set to 1.
-pipe 1 | 0	Pipelines registers located outside of a multiplier module back into the module to create pipeline stages. Default is 1 for ON. See *Pipelining* , on page 434 in the *User Guide* for a procedure.
-resource_sharing 1|0	Enables or disables resource sharing. See *Sharing Resources* , on page 454 in the *User Guide* for information about using 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.
-retiming 1 | 0	When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). Enabling -retiming also enables -pipe. See *Retiming* , on page 438 in the *User Guide* 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 377 in the *User Guide* for more information.
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle* , on page 203.
-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 328, 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 329, for details.
- seqshift_no_replicate 1 | 0	Determines whether timing-driven register replication happens for a sequential shift circuit. By default, the value is set to 0 which means register replications take place.

Technology Keywords

The following table lists the -technology keyword for all Lattice technologies:

Family	Valid -technology Keywords
iCE40	SBTiCE40
iCE40LM	SBTiCE40LM
LatticeECP/EC	ECP5U, ECP5UM, LATTICE-ECP3, LATTICE-ECP2S, LATTICE-ECP2MS, LATTICE-ECP2M, LATTICE-ECP2, LATTICE-ECP, LATTICE-EC
LatticeSC LatticeSCM	LATTICE-SC, LATTICE-SCM
LatticeXP	LATTICE-XP2, LATTICE-XP,
MACHXO/ Platform Manager	MACHXO, MACHXO2, Platform_Manager, Platform_Manager_2

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 *Attributes and Directives Summary*, on page 13.

Attribute/Directive	**Description**
black_box_pad_pin	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
full_case	Specifies that a Verilog case statement has covered all possible cases.
loc	Specifies pin, register, and bus locations for Lattice I/Os.
loop_limit	Specifies a loop iteration limit for for loops.
parallel_case	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
pragma translate_off/pragma translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
syn_allow_retiming	Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.
syn_allowed_resources	Specifies the maximum number of technology-specific resources available for use in a design.
syn_black_box	Defines a black box for synthesis.
syn_direct_enable	Identifies which signal to use as the enable input to an enable flip-flop, when multiple candidates are possible. (LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families)

Attribute/Directive	Description
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_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.
syn_insert_buffer	Inserts a clock buffer according to the vendor-specified value.
syn_insert_pad	Inserts I/O buffers to either ports or nets.

Attribute/Directive	Description
syn_isclock	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_keep	Prevents the internal signal from being removed during synthesis and optimization.
syn_looplimit	Specifies a loop iteration limit for while loops.
syn_maxfan	Sets a fanout limit for an individual input port or register output. (LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families)
syn_multstyle	Determines how multipliers are implemented. (LatticeECP, LatticeSC, LatticeXP, MachXO, and Platform Manager families)
syn_netlist_hierarchy	Determines if the EDIF output netlist is flat or hierarchical. (LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families)
syn_noarrayports	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals. (LatticeECP/EC, LatticeXP, LatticeSC/SCM, MachXO, and Platform Manager families)
syn_noclockbuf	Disables automatic clock buffer insertion.
syn_noclockpad	Infers SB_GB and SB_IO instead of SB_GB_IO on clock nets.
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.
syn_pipeline	Specifies that registers be moved into multipliers and ROMs in order to improve frequency.
syn_preserve	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
syn_probe	Inserts probe points for testing and debugging the internal signals of a design.

Attribute/Directive	Description
syn_ramstyle	Determines the way in which RAMs are implemented for the LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families.
syn_reduce_controlset_size	Controls the minimum size of the unique control-set on which control-set optimizations can occur.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
syn_replicate	Disables replication.
syn_romstyle	Determines the way in which ROMs are implemented for the iCE40, LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families.
syn_sharing	Specifies resource sharing of operators.
syn_shift_resetphase	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.
syn_srlstyle	Determines how sequential shift components are implemented.
syn_state_machine	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n>	Defines timing clock to output delay through the black box. The *n* indicates a value between 1 and 10.
syn_tpd<n>	Specifies timing propagation for combinational delay through the black box. The *n* indicates a value between 1 and 10.
syn_tristate	Specifies that a black-box pin is a tristate pin.
syn_tsu<n>	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.
syn_use_carry_chain	Use this attribute to turn on or off carry chain implementation for arithmetic and combinational operators.

Attribute/Directive	Description
syn_useenables	For CPLD technologies, generates register instances with clock enable pins for selective clock-enable inference.
syn_useioff	Packs flip-flops in the I/O ring to improve input/output path timing in LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families.
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

Index

Symbols

_ta.srm file 157
.adc file 146
.fse file 154
.info file 154
.ini file 146
.lpf file 323
.prj file 146
.sap
  annotated properties for analyst 155
.sar file 156
.sdc file 147
.srd file 156
.srm file 156
.srr file 160
  watching selected information 40
.srs file 156
  initial values (Verilog) 214
.sv file 147
  SystemVerilog source file 148
.ta file
  See timing report file 157
.v file 147
.vhd file 147
.vhm file 159
.vm file 159

A

adc file (analysis design constraint) 146
adder
  SYNCore 262
Allow Docking command 41
Alt key, selecting columns in Text Editor 58
analysis design constraint file (.adc) 146
Analyst toolbar 71
annotated properties for analyst
  .sap 155
  .timing annotated properties (.tap) 158
archive file (.sar) 156
arrow keys, selecting objects in Hierarchy Browser 112
arrow pointers for push and pop 110
asynchronous clock report
  description 172
attributes
  inferring RAM 186
Attributes demo 60
auto constraints 143
  Maximize option 85

B

batch mode 20
black boxes
  Lattice 309
block RAM
  dual-port RAM examples 202
  inferring 189
  modes 185
  NO_CHANGE mode example 198
  READ_FIRST mode example 197
  single-port RAM examples 199
  types 185
  WRITE_FIRST mode example 195
block RAMs
  Lattice 298
buttons and options, Project view 84

C

callback functions, customizing flow 280
cck.rpt file (constraint checking report) 154
check boxes, Project view 84
clock buffering report, log file (.srr) 162
clock frequency goals, tradeoffs using different 282
clock groups
  Clock Relationships (timing report) 168

clock pin drivers, selecting all 53
clock relationships, timing report 168
clock report
  asynchronous 165
  detailed 170
Clock Tree, HDL Analyst tool 53
clocks
  asynchronous report 172
  declared clock 167
  defining 53
  derived clock 168
  inferred clock 167
  system clock 168
color coding
  Text Editor 57
commands
  Tcl hooks 280
commenting out code (Text Editor) 58
compile points
  Lattice 328
compiler report, log file (.srr) 161
compilers 21
Constraint Check command 174
constraint checking report 174
constraint files 127
  .sdc 147
  automatic. *See* auto constraints
  fdc and sdc precedence order 131
  forward-annotation, Lattice 322
  tcl script examples 283
constraint files (.sdc)
  creating 70
constraint priority 131
constraints
  auto constraints. *See* auto constraints
  non-DC 137
  priority 131
  report file 174
  styles 129
  types 126
context help editor 58
context of filtered schematic, displaying 117
context sensitive help
  using the F1 key 17
copying
  for pasting 77
counter compiler
  SYNCore 274
critical paths 121
  analyzing 122
  finding 122
cross-clock paths, timing analysis 169
cross-hair mouse pointer 67
crossprobing 102
  definition 102
Ctrl key
  avoiding docking 69
  multiple selection 66
  zooming using the mouse wheel 68
customization
  callback functions 280
cutting (for pasting) 70

D

declared clock 167
deleting
  See removing
derived clock 168
design flow
  customizing with callback functions 280
Design Hierarchy view (hierarchichal project management) 29
Design Hierarchy view (hierarchichal project management) icons 30
design size, schematic sheet
  setting 105
device options
  LatticeECP2 and LatticeECP/EC 334
directory
  examples delivered with synthesis tool 22
disable I/O insertion (Lattice) 327
Dissolve Instances command 119
docking 41
  avoiding 69
docking GUI entities
  toolbar 69
dual-port RAM examples 202

E

editor view
  context help 58
encoding
  state machine
    FSM Compiler 62
examples

Interactive Attribute Examples 60
examples delivered with synthesis tool, directory 22

F

failures, timing (definition) 123
fanouts (Lattice) 327
fdc
constraint priority 131
precedence over sdc 131
fdc constraints 132
generation process 130
fdc file
relationship with other constraint files 127
FIFO compiler
SYNCore 222
FIFO flags
empty/almost empty 230
full/almost full 229
handshaking 231
programmable 232
programmable empty 236
programmable full 233
files
.adc 146
.fdc 146
.fse 154
.info 154
.ini 146
.lpf 323
.prj 18, 146
.sar 156
.sdc 147
.srm 156, 157
.srr 160
watching selected information 40
.srs 156
.ta 157
.v 147
.vhd 147
.vhm 159
.vm 159
compiler output (.srs) 156
constraint (.adc) 146
constraint (.sdc) 147
creating 70
customized timing report (.ta) 157
design component info (.info) 154
initialization (.ini) 146
log (.srr) 160
watching selected information 40
mapper output (.srm) 156, 157
output
See output files
project (.prj) 18, 146
RTL view (.srs) 156
srr 160
watching selected information 40
state machine encoding (.fse) 154
Synplicity archive file (.sar) 156
synthesis output 154
Technology view (.srm) 156, 157
Verilog (.v) 147
VHDL (.vhd) 147
files for synthesis 146
filtered schematic
compared with unfiltered 88
filtering 115
commands 115
compared with flattening 120
FSM states and transitions 55
paths from pins or ports 124
filtering critical paths 122
finding
critical paths 122
information on synthesis tool 24
GUI 17
finite state machines
See state machines
Flatten Current Schematic command 119
Flatten Schematic command 119
flattening
commands 117
compared with filtering 120
selected instances 118
Float command
Watch window popup menu 41
floating
toolbar 69
floating toolbar popup menu 69
forward annotation
initial values 214
Forward Annotation of Initial Values
Verilog 214
forward-annotation
ispLEVER 323
Lattice constraints 323

forward-annotation (Lattice) 322
frequency
  cross-clock paths 169
Frequency (Mhz) option, Project view 85
fse file 154
FSM Compiler option, Project view 85
FSM Compiler, enabling and disabling
  globally
    with GUI 85, 331
  locally, for specific register 63
FSM encoding file (.sfe) 154
FSM toolbar 74
FSM Viewer 54
FSMs (finite state machines)
  See state machines

G

gated clocks (Lattice iCE60/iCE40) 332
generated clocks (Lattice) 333
generic technology library 151
global buffer promotion
  iCE40 321
graphical user interface (GUI), overview 25
GSR resource (Lattice) 328
GTECH library. *See* generic technology library
gtech.v library 151
gui
  synthesis software 15
GUI (graphical user interface), overview 25

H

HDL Analyst tool 87
  accessing commands 89
  analyzing critical paths 121
  Clock Tree 53
  crossprobing 102
  filtering designs 115
  finding objects 99
  hierarchical instances. *See* hierarchical instances
  object information 90
  preferences 105
  push/pop mode 108
  ROM table viewer 215
  schematic sheet size 105
  schematics, filtering 115
  schematics, multiple-sheet 105
  status bar information 90
  title bar information 105
HDL Analyst toolbar
  See Analyst toolbar
HDL Analyst views 88
  See also RTL view, Technology view
HDL files, creating 70
header, timing report 166
help
  online
    accessing 17
hidden hierarchical instances 95
  are not flattened 119
Hide command
  floating toolbar popup menu 69
  Log Watch window popup menu 41
  Tcl Window popup menu 44
hierarchical instances 93
  compared with primitive 92
  display in HDL Analyst 93
  hidden 95
  opaque 93
  transparent 93
hierarchical project management views 29
hierarchical schematic sheet, definition 105
hierarchy
  flattening
    compared with filtering 120
  pushing and popping 108
  schematic sheets 105
Hierarchy Browser 111
  changing size in view 49
  Clock Tree 53
  finding schematic objects 99
  moving between objects 53
  RTL view 49
  symbols (legend) 53
  Technology view 51
  trees of objects 52

I

IEEE 1364 Verilog 95 standard 149
Implementation Directory 34
Implementation Results 34
indenting a block of text 57
indenting text (Text Editor) 57
inferred clock 167

info file (design component info) 154
ini file 146
initial value data file
  Verilog 212
Initial Values
  forward annotation 214
initial values
  $readmemb 208
  $readmemh 208
  Lattice ICE65/iCE40 314
initial values (Verilog)
  netlist file (.srs) 214
initialization file (.ini) 146
input files 146
  .adc 146
  .ini 146
  .sdc 147
  .sv 148
  .v 147, 148
  .vhd 148
  vhd 147
inserting
  bookmarks (Text Editor) 57
instances
  hierarchical
    dissolving 113
    making transparent 113
  hierarchical. *See* hierarchical instances
  primitive. *See* primitive instances
Interactive Attribute Examples 60
interface information, timing report 170
IP encryption
  Lattice 289
IP encryption (Lattice) 289
isolating paths from pins or ports 124

K

keyboard shortcuts 76
  arrow keys (Hierarchy Browser) 112
keyword completion, Text Editor 57
keywords
  completing in Text Editor 57

L

latches
  in timing analysis 121
Lattice
  attributes and directives 352
  black boxes 309
  Block RAM Support 298
  disable I/O insertion 327
  forward annotation 322
  forward-annotation constraints 323
  generated clocks 333
  GSR resource 328
  IP encryption 289
  macros 309
  pipelining 331
  product families 288
  reports 323
  retiming 332
  updating compile points 328
Lattice device options
  LatticeSC/SCM and LatticeXP 338
  MachXO 341
Lattice iCE65/iCE40
  gated clocks 332
  initial values 314
  set_option synthesis options 347
Lattice iCE65/iCE40 options 346
Lattice options
  LatticeECP2 and LatticeECP/EC 335
  MachXO 342
LatticeECP/EC and Later
  set_option synthesis options 337
LatticeSC/SCM and LatticeXP2/XP
  set_option synthesis options 341
legacy sdc file. *See* sdc files, difference between legacy and Synopsys standard
libraries
  general technology 150
  macro, built-in 148
  technology-independent 150
  VHDL
    attributes and constraints 148
license
  specifying in batch mode 20
linkerlog file 154
log file (.srr) 160
  watching selected information 40
log file report 160
  clock buffering 162
  compiler 161
  mapper 161
  net buffering 162
  resource usage 163

retiming 164
summary of compile points 163
timing 163
Log Watch Configuration dialog box 42
Log Watch window 40
Output Windows 48
positioning commands 41

M

MachXO and Platform
set_option synthesis options 344
macros
libraries 148
macros (Lattice) 309
mapper output file (.srm) 156, 157
mapper report
log file (.srr) 161
margin, slack 122
message viewer
description 44
Messages Tab 44
mouse button operations 66
mouse operations 64
Mouse Stroke Tutor 65
mouse wheel operations 68
Move command
floating toolbar window 69
Log Watch window popup menu 41
Tcl window popup menu 44
moving between objects in the Hierarchy Browser 53
moving GUI entities
toolbar 69
multiple target technologies, synthesizing with Tcl script 282
multiple-sheet schematics 105
multisheet schematics
transparent hierarchical instances 107

N

navigating
among hierarchical levels
by pushing and popping 108
with the Hierarchy Browser 111
among the sheets of a schematic 105
nesting design details (display) 113
net buffering report, log file 162
netlist file 159
initial values (Verilog) 214

O

object information
status bar, HDL Analyst tool 90
viewing in HDL Analyst tool 90
objects
crossprobing 102
dissolving 113
making transparent 113
objects, schematic
See schematic objects
Online help
F1 key 17
online help
accessing 17
opaque hierarchical instances 93
are not flattened 119
optimization
state machines 62
options
Lattice iCE65/iCE40 346
LatticeSC/SCM and LatticeXP 339
Project view 84
Frequency (Mhz) 85
FSM Compiler 85
Pipelining 86
Resource Sharing 86
Retiming 86
setting with set_option Tcl command 283
output files 154
.info 154
.sar 156
.srm 156, 157
.srr 160
watching selected information 40
.srs 156
.ta 157
.vhm 159
.vm 159
netlist 159
See also files
Output Windows 48
Overview of the Synopsys FPGA Synthesis Tools 14

P

partitioning of schematics into sheets 105
pasting 70
performance summary, timing report 166
pins
  displaying
    on transparent instances 97
  displaying on technology-specific primitives 98
  isolating paths from 124
pipelining
  option 86
pipelining (Lattice) 331
Pipelining option, Project view 86
pointers, mouse
  cross-hairs 67
  push/pop arrows 110
popping up design hierarchy 108
popup menus
  floating toolbar 69
  Log Watch window 41, 42
  Log Watch window positioning 41
  Tcl window 44
precedence of constraint files 131
preferences
  HDL Analyst tool 105
primitive instances 92
primitives
  pin names in Technology view 98
prj file 18, 146
Process View 36
project files (.prj) 18, 146
Project Files view 29
project results
  Implementation Directory 34
  Process View 36
  Project Status View 31
Project Results View 31
Project Status View 31
Project toolbar 69
Project view 26
  buttons and options 84
  options 84
  Synplify Pro 26
Project window 26
project_name_cck.rpt file 174
push/pop mode, HDL Analyst tool 108

R

RAM inference 185
  using attributes 186
RAMs
  inferring block RAM 189
  initial values (Verilog) 208
RAMs, inferring
  advantages 184
reference manual, role in document set 13
removing
  bookmark (Text Editor) 57
  window (view) 69
reports
  constraint checking (cck.rpt) 174
  resource usage, Lattice 323
reports (Lattice) 323
Resource Sharing option, Project view 86
resource usage report, log file 163
retiming
  report, log file 164
retiming (Lattice) 332
Retiming option, Project view 86
ROM compiler
  SYNCore 256
ROM inference examples 215
ROM initialization
  with rom.info file 218
  with Verilog generate block 219
rom.info file 215
RTL view 49
  displaying 72
  file (.srs) 156

S

schematic objects
  crossprobing 102
  definition 90
  dissolving 113
  finding 99
  making transparent 113
  status bar information 90
schematic sheets 105
  hierarchical (definition) 105
  navigating among 105
  setting size 105
schematics
  configuring amount of logic on a sheet 105

crossprobing 102
filtered 88
filtering commands 115
flattening compared with filtering 120
flattening selectively 118
hierarchical (definition) 105
multiple-sheet 105
multiple-sheet. *See also* schematic sheets
object information 90
partitioning into sheets 105
sheet connectors 91
sheets
navigating among 105
size, setting 105
size in view, changing 49
unfiltered 88
unfiltering 117
SCOPE
for legacy sdc 134
sdc
fdc precedence 131
SCOPE for legacy files 134
sdc file
difference between legacy and Synopsys standard 129
sdc2fdc utility 135
selecting
text column (Text Editor) 58
selecting multiple objects using the Ctrl key 66
set_rtl_ff_names 137
sheet connectors 91
Shift key 69
shortcuts
keyboard
See keyboard shortcuts
single-port RAM examples 199
slack
cross-clock paths 169
defined 167
margin
definition 123
setting 122
source files
See also files
creating 70
srd file 156
srm file 156, 157
srr file 160
watching selected information 40
srs file 156
initial values (Verilog) 214
standards, supported
Verilog 149
VHDL 148
starting Synplify Pro 19
state machines
encoding
displaying 55
FSM Compiler 62
encoding file (.fse) 154
filtering states and transitions 55
optimization 62
state encoding, displaying 55
status bar information, HDL Analyst tool 90
structural netlist file (.vhm) 159
structural netlist file (.vm) 159
subtractor
SYNCore 262
summary of compile points report
log file (.srr) 163
supported standards
Verilog 149
VHDL 148
symbols
Hierarchy Browser (legend) 53
SYN_TCL_HOOKS variable 280
SYNCore
adder/subtractor 262
byte-enable RAM compiler
byte-enable RAM compiler
SYNCore 250
counter compiler 274
FIFO compiler 222
RAM compiler
RAM compiler
SYNCore 240
ROM compiler 256
SYNCore adder/subtractor
adders 263
dynamic adder/subtractor 269
functional description 262
subtractors 266
SYNCore FIFOs
definition 222
parameter definitions 227
port list 225

read operations 224
status flags 229
write operations 223
SYNCore ROMs
clock latency 260
dual-port read 258
parameter list 259
single-port read 257
synhooks.tcl file 280
Synopsys FPGA Synthesis Tools
overview 14
Synopsys standard sdc file. *See* sdc files, difference between legacy and Synopsys standard
Synplicity directory (ini file) 146
Synplify Pro tool
Project view 26
user interface 15
synplify_pro command-line command 20
synthesis
log file (.srr) 160
watching selected information 40
synthesis software
flow 20
gui 15
system clock 168
SystemVerilog keywords
context help 58

T

ta file (customized timing report) 157
Tcl commands
constraint files 133
pasting 44
syntax for Tcl hooks 280
Tcl Script window
Output Windows 48
Tcl scripts
examples 282
Tcl shell command
sdc2fdc 135
Tcl window
popup menu commands 44
popup menus 44
Technology view 50
displaying 72
file (.srm) 156, 157
Text Editor
features 57
indenting a block of text 57
opening 56
selecting text column 58
view 56
text editor
completing keywords 57
Text Editor view 56
timing analysis of critical paths (HDL Analyst tool) 121
timing analyst
cross-clock paths 169
timing annotated properties (.tap) 158
timing constraints
See also FPGA timing constraints
See constraints
timing failures, definition 123
timing report 165
clock relationships 168
customized (.ta file) 157
detailed clock report 170
file (.ta) 157
header 166
interface information 170
performance summary 166
timing reports
asynchronous clocks 172
log file (.srr) 163
title bar information, HDL Analyst tool 105
toolbars 69
FSM 74
moving and docking 69
transparent hierarchical instances 94
lower-level logic on multiple sheets 107
operations resulting in 114
pins and pin names 97
trees of objects, Hierarchy Browser 52
trees, browser, collapsing and expanding 53

U

unfiltered schematic, compared with filtered 88
unfiltering schematic 117
user interface
Synplify Pro tool 15
user interface, overview 25
using the mouse 64
utilities

sdc2fdc 135

V

v file 147
vendor technologies
Lattice 287
vendor-specific Tcl commands 280
Verilog
Forward Annotation of Initial Values 214
generic technology library 151
initial value data file 212
initial values for RAMs 208
netlist file 159
ROM inference 215
source files (.v) 147
structural netlist file (.vm) 159
supported standards 149
Verilog 2001 149
Verilog 95 149
Verilog source file (.v) 148
vhd file 147
vhd source file 147
VHDL
libraries
attributes, supplied with synthesis tool 148
source files (.vhd) 147
structural netlist file (.vhm) 159
supported standards 148
VHDL source file (.vhd) 148
vhm file 159
views 39
FSM 54
Project 26
removing 69
RTL 49
Technology 50
vm file 159

W

Watch Window. *See* Log Watch window
window
Project 26
windows 39
closing 78
log watch 40
removing 69

Z

zoom
using the mouse wheel and Ctrl key 68