

Synopsys Synplify Pro for Microsemi Edition

Reference Manual

November 2016



Copyright Notice and Proprietary Information

© 2016 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

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

Third-Party Links

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

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

November 2016

Contents

Chapter 1: Product Overview

Synopsys FPGA and Prototyping Products	14
FPGA Implementation Tools	14
Identify Tool Set	16
Symphony Model Compiler	16
Rapid Prototyping	17
Overview of the Synthesis Tools	18
Common Features	18
BEST Algorithms	19
Graphic User Interface	19
Projects and Implementations	22
Starting the Synthesis Tool	23
Logic Synthesis Overview	24
Synthesizing Your Design	25
Getting Help	28

Chapter 2: User Interface Overview

The Project View	30
Project Management View	32
The Project Results View	34
Project Status Tab	34
Implementation Directory	39
Process View	40
Other Windows and Views	43
Dockable GUI Entities	44
Watch Window	44
Tcl Script and Messages Windows	47
Tcl Script Window	48
Message Viewer	48

Output Windows (Tcl Script and Watch Windows)	52
RTL View	53
Technology View	54
Hierarchy Browser	56
FSM Viewer Window	58
Text Editor View	60
Context Help Editor Window	63
Interactive Attribute Examples	64
Search SolvNet	66
FSM Compiler	67
When to Use FSM Compiler	68
Where to Use FSM Compiler (Global and Local Use)	68
FSM Explorer	69
Using the Mouse	69
Mouse Operation Terminology	70
Using Mouse Strokes	70
Using the Mouse Buttons	72
Using the Mouse Wheel	74
User Interface Preferences	74
Managing Views	75
Toolbars	76
Project Toolbar	76
Analyst Toolbar	78
Text Editor Toolbar	80
FSM Viewer Toolbar	81
Tools Toolbar	82
Keyboard Shortcuts	84
Buttons and Options	92

Chapter 3: HDL Analyst Tool

HDL Analyst Views and Commands	96
Filtered and Unfiltered Schematic Views	96
Accessing HDL Analyst Commands	97
Schematic Objects and Their Display	98
Object Information	98
Sheet Connectors	99
Primitive and Hierarchical Instances	100
Transparent and Opaque Display of Hierarchical Instances	101

Hidden Hierarchical Instances	103
Schematic Display	103
Basic Operations on Schematic Objects	107
Finding Schematic Objects	107
Selecting and Unselecting Schematic Objects	108
Crossprobing Objects	109
Dragging and Dropping Objects	111
Multiple-sheet Schematics	112
Controlling the Amount of Logic on a Sheet	112
Navigating Among Schematic Sheets	112
Multiple Sheets for Transparent Instance Details	114
Exploring Design Hierarchy	115
Pushing and Popping Hierarchical Levels	115
Navigating With a Hierarchy Browser	118
Looking Inside Hierarchical Instances	120
Filtering and Flattening Schematics	122
Commands That Result in Filtered Schematics	122
Combined Filtering Operations	123
Returning to The Unfiltered Schematic	123
Commands That Flatten Schematics	124
Selective Flattening	125
Filtering Compared to Flattening	126
Timing Information and Critical Paths	128
Timing Reports	128
Critical Paths and the Slack Margin Parameter	129
Examining Critical Path Schematics	130

Chapter 4: Constraints

Constraint Types	134
Constraint Files	135
Timing Constraints	137
FDC Constraints	140
Methods for Creating Constraints	141
Constraint Translation	143
sdc2fdc Conversion	143

Constraint Checking	146
Database Object Search	148
Forward Annotation	149
Auto Constraints	149

Chapter 5: SCOPE Constraints Editor

SCOPE User Interface	152
SCOPE Tabs	153
Clocks	153
Generated Clocks	159
Collections	161
Inputs/Outputs	163
Registers	167
Delay Paths	168
Attributes	170
I/O Standards	171
Compile Points	173
TCL View	176
Industry I/O Standards	178
Industry I/O Standards	179
Delay Path Timing Exceptions	182
Multicycle Paths	182
False Paths	185
Specifying From, To, and Through Points	187
Timing Exceptions Object Types	187
From/To Points	187
Through Points	189
Product of Sums Interface	190
Clocks as From/To Points	192
Conflict Resolution for Timing Exceptions	194
SCOPE User Interface (Legacy)	198

Chapter 6: Constraint Syntax

FPGA Timing Constraints	200
create_clock	202
create_generated_clock	204
reset_path	207

set_clock_groups	209
set_clock_latency	213
set_clock_route_delay	215
set_clock_uncertainty	216
set_false_path	218
set_input_delay	221
set_max_delay	223
set_multicycle_path	226
set_output_delay	230
set_reg_input_delay	233
set_reg_output_delay	234
Naming Rule Syntax Commands	234
Design Constraints	237
define_compile_point	238
define_current_design	239
define_io_standard	240

Chapter 7: Input and Result Files

Input Files	242
HDL Source Files	243
Libraries	245
Open Verification Library (Verilog)	246
The Generic Technology Library	246
ASIC Library Files	247
Output Files	249
Log File	254
Timing Reports	259
Timing Report Header	260
Performance Summary	260
Clock Relationships	262
Interface Information	263
Detailed Clock Report	264
Asynchronous Clock Report	266

Hierarchical Area Report	267
Constraint Checking Report	268

Chapter 8: RAM and ROM Inference

Guidelines and Support for RAM Inference	278
Block RAM Examples	279
Block RAM Mode Examples	279
Single-Port Block RAM Examples	283
Single-Port RAM with Read Address Registered Example	283
Single-Port RAM with RAM Output Registered Examples	284
Dual-Port Block RAM Examples	286
True Dual-Port RAM Examples	288
Initial Values for RAMs	292
Example 1: RAM Initialization	292
Example 2: Cross-Module Referencing for RAM Initialization	293
Initialization Data File	294
Forward Annotation of Initial Values	297
RAM Instantiation with SYNCORE	297
ROM Inference	298

Chapter 9: IP and Encryption Tools

SYNCore FIFO Compiler	304
Synchronous FIFOs	304
FIFO Read and Write Operations	305
FIFO Ports	307
FIFO Parameters	309
FIFO Status Flags	311
FIFO Programmable Flags	314
SYNCore RAM Compiler	321
Single-Port Memories	321
Dual-Port Memories	323
Read/Write Timing Sequences	328
SYNCore Byte-Enable RAM Compiler	331
Functional Overview	331
Read/Write Timing Sequences	332
Parameter List	335
SYNCore ROM Compiler	336
Functional Overview	336

Single-Port Read Operation	338
Dual-Port Read Operation	338
Parameter List	339
Clock Latency	340
SYNCore Adder/Subtractor Compiler	342
Functional Description	342
Adder	343
Subtractor	346
Dynamic Adder/Subtractor	349
SYNCore Counter Compiler	354
Functional Overview	354
UP Counter Operation	355
Down Counter Operation	355
Dynamic Counter Operation	356
Encryption Scripts	359
Encryption and Decryption Methodologies	359
The encryptP1735 Script	360
The encryptIP Script	364

Chapter 10: Scripts

<i>synhooks</i> File Syntax	370
Tcl Script Examples	372
Using Target Technologies	372
Different Clock Frequency Goals	372
Setting Options and Timing Constraints	373

Appendix A: Designing with Microsemi

Basic Support for Microsemi Designs	378
Microsemi Device-specific Support	378
Microsemi Features	379
Synthesis Constraints and Attributes for Microsemi	379
Microsemi Components	381
Macros and Black Boxes in Microsemi Designs	381
DSP Block Inference	383
Microsemi RAM Implementations	387
Instantiating RAMs with SYNCORE	405
Control Signals Extraction for Registers (SLE)	406
Output Files and Forward-annotation for Microsemi	407
VM Flow Support	407

Forward-annotating Constraints for Placement and Routing	408
Synthesis Reports	409
Optimizations for Microsemi Designs	410
The syn_maxfan Attribute in Microsemi Designs	410
Promote Global Buffer Threshold	411
I/O Insertion	412
Number of Critical Paths	413
Retiming	413
Update Compile Point Timing Data Option	413
Operating Condition Device Option	415
Radiation-tolerant Applications	417
Integration with Microsemi Tools and Flows	419
Compile Point Synthesis	419
Incremental Synthesis Flow	420
Microsemi Place-and-Route Tools	420
Microsemi Device Mapping Options	421
Microsemi Tcl set_option Command Options	423
Microsemi Attribute and Directive Summary	426

CHAPTER 1

Product Overview

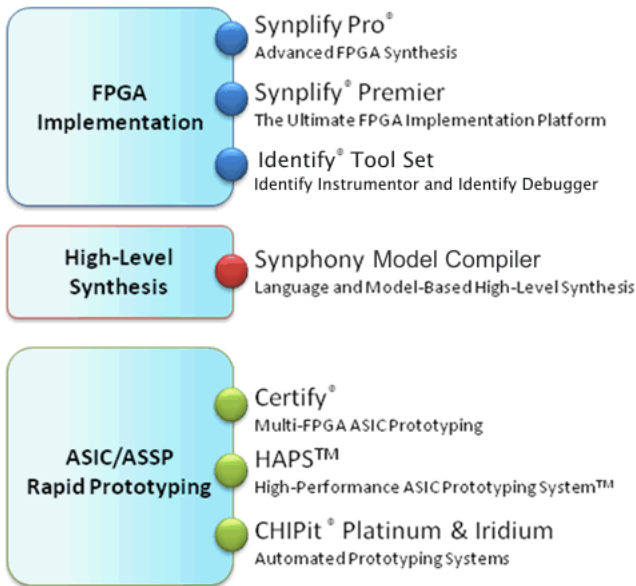
This document is part of a set that includes reference and procedural information for the Synplify Pro[®] synthesis tool. The reference manual details the synthesis tool user interface, commands, and features. The user guide contains “how-to” information, emphasizing user tasks, procedures, design flows,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

The following provide an introduction to the synthesis tools.

- [Synopsys FPGA and Prototyping Products, on page 14](#)
- [Overview of the Synthesis Tools, on page 18](#)
- [Starting the Synthesis Tool, on page 23](#)
- [Logic Synthesis Overview, on page 24](#)
- [Getting Help, on page 28](#)

Synopsys FPGA and Prototyping Products

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



FPGA Implementation Tools

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

Synplify Pro Product

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

Synplify Premier Product

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

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

- Fast and Advanced Synthesis Modes
- Design Planning (Optional)
- DesignWare Support
- Power Switching Activity (SAIF Generation)

Identify Tool Set

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

Synphony Model Compiler

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

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

Rapid Prototyping

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

Certify Product

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

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

Overview of the Synthesis Tools

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

- [Common Features, on page 18](#)
- [BEST Algorithms, on page 19](#)
- [Graphic User Interface, on page 19](#)
- [Projects and Implementations, on page 22](#)

Common Features

The Synopsys FPGA synthesis tool has 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 53](#), [Technology View, on page 54](#), and [Analyzing With the HDL Analyst Tool, on page 298](#) 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 60](#).
- 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 152](#).
- FSM Compiler, a symbolic compiler that performs advanced finite state machine (FSM) optimizations. See [FSM Compiler, on page 67](#).
- Integration with the Identify RTL Debugger.
- FSM Explorer, which tries different state machine optimizations before picking the best implementation. See [FSM Explorer, on page 69](#).
- The FSM Viewer, for viewing state transitions in detail. See [FSM Viewer Window, on page 58](#).
- The Tcl window, a command line interface for running TCL scripts. See [Tcl Script Window, on page 48](#).

- The Timing Analyst window, which allows you to generate timing schematics and reports for specified paths for point-to-point timing analysis.
- Place-and-Route implementation(s) to automatically run placement and routing after synthesis. You can run place-and-route from within the tool or in batch mode. This feature is supported for the latest Microsemi technologies (see [Running P&R Automatically after Synthesis](#), on page 550 in the *User Guide*).
- Other special windows, or *views*, for analyzing your design, including the Watch Window and Message Viewer (see [The Project View](#), on page 30).
- Certain optimizations, like retiming, are only available with this tool.
- Advanced analysis features like crossprobing and probe point insertion.

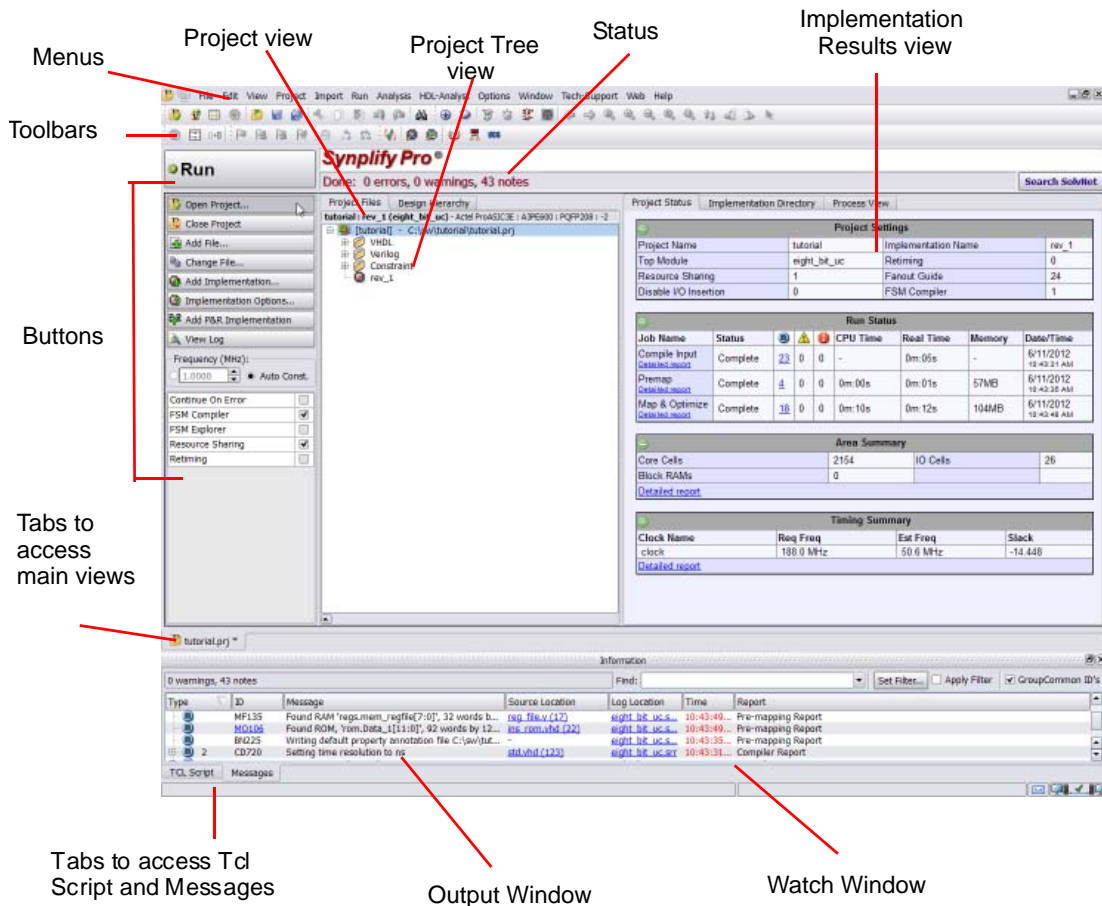
BEST Algorithms

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 figures show the graphical user interfaces for the Synplify Pro tool.



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 30
RTL view	RTL View, on page 53
Technology view	Technology View, on page 54
Text Editor view	Text Editor View, on page 60
FSM Viewer window	FSM Viewer Window, on page 58
Tcl window	Tcl Script Window, on page 48
Watch Window	Watch Window, on page 44
SCOPE spreadsheet	SCOPE User Interface, on page 152
Other views and windows	The Project View, on page 30
Menu commands and their dialog boxes	Chapter 4, <i>User Interface Commands</i>
Toolbars	Toolbars, on page 76
Buttons	Buttons and Options, on page 92
Context-sensitive popup menus and their dialog boxes	Chapter 5, <i>GUI Popup Menu Commands</i>
Online help	Use the F1 keyboard shortcut or click the Help button in a dialog box. See Help Menu, on page 331 , for more information.

Projects and Implementations

Projects and implementations are available for all synthesis tools.

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

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

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

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 `synplify_pro` at the system prompt.

For information about using the synthesis tool in batch mode, see [Starting the Tool in Batch Mode, on page 23](#).

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]
```

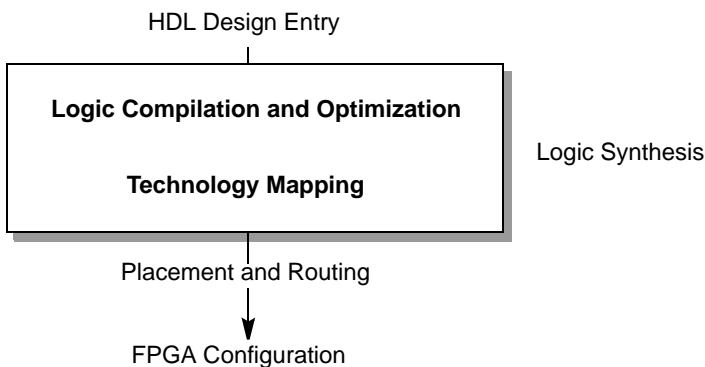
In the syntax statement, *toolName* is `synplify_pro`.

For complete syntax details, refer to [synplify_pro, on page 86](#) in the *Command Reference*.

Logic Synthesis Overview

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

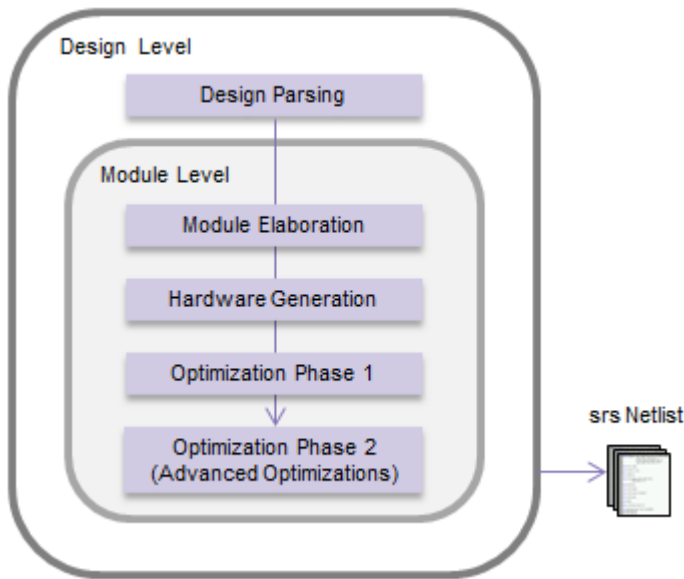
- Logic compilation (HDL language synthesis) and optimization
- Technology mapping



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 improving 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 in formats such as EDIF, 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 153](#), 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 the Tcl window (View->Tcl Window). The log file contains reports with information on timing, usage, and net buffering.

If synthesis is successful, you see the message Done! or Done (warnings). If processing stops because of syntax errors or other design problems, you see the message Errors! displayed, along with the error status in the log file and 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 SolvNet 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...	<i>User Guide</i> and various application notes available on the Synopsys support website
Flow information	<i>User Guide</i> and various application notes available on the Synopsys SolvNet support website
FPGA Implementation Tools	Synopsys Web Page (Web->FPGA Implementation Tools menu command from within the software)
Synthesis features	<i>User Guide</i> and <i>Reference Manual</i>
Language and syntax	<i>Reference Manual</i>
Attributes and directives	<i>Attribute Reference Manual</i>
Tcl language	Online help (Help->Tcl Help)
Synthesis Tcl commands	<i>Command Reference Manual</i> or type help followed by the command name in the Tcl window
Using tool-specific features and attributes	<i>User Guide</i>
Error and warning messages	Click the message ID code

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 30](#)
- [The Project Results View, on page 34](#)
- [Other Windows and Views, on page 43](#)
- [FSM Compiler, on page 67](#)
- [FSM Explorer, on page 69](#)
- [Using the Mouse, on page 69](#)
- [User Interface Preferences, on page 74](#)
- [Toolbars, on page 76](#)
- [Keyboard Shortcuts, on page 84](#)
- [Buttons and Options, on page 92](#)

The Project View

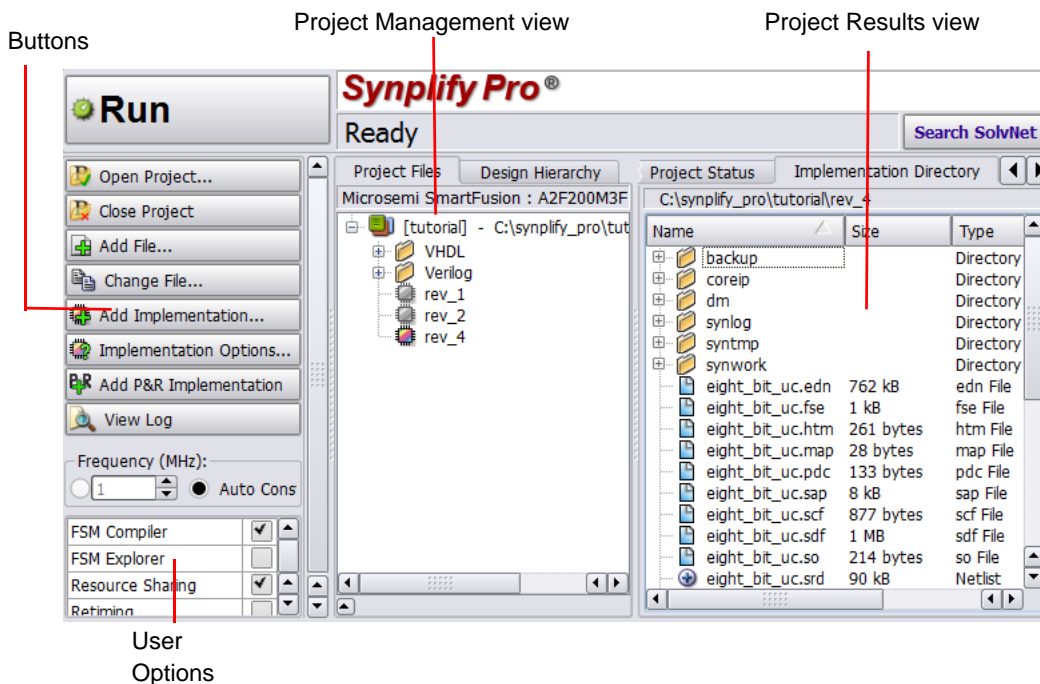
The Project View is the main interface to the tool. The Project view consists of a Project Management View on the left and a Project Results View on the right. See [Multiple Pane Project View, on page 30](#) for an overview.

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. You can use it to manage and synthesize hierarchical designs. The Project Results view is on the right.

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

- [Project Management View, on page 32](#)
- [The Project Results View](#)



The Project view has the following main parts:

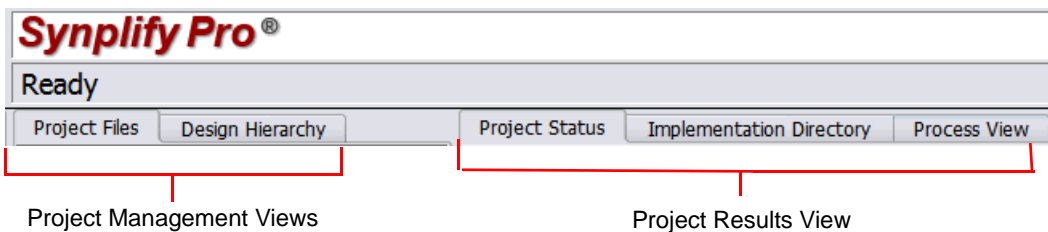
Project View Interface	Description
Status	Displays the tool name or the current status of the synthesis job that is running. Clicking in this area displays additional information about the current job.
Buttons and options	Allow immediate access to some of the more common commands. See Buttons and Options, on page 92 for details.
Hierarchical Project Management view	<p>Lists the projects and implementations, and their associated HDL source files and constraint files. There are two tabs with different views to facilitate working with hierarchical designs:</p> <ul style="list-style-type: none"> • Project Files tab • Design Hierarchy tab
Implementation Results view	<p>Lists the result of the synthesis runs for the implementations of your design. You can only view one set of implementation results at a time. Click an implementation in the Project view to make it active and view its result files.</p> <p>The Project Results view includes the following:</p> <ul style="list-style-type: none"> • Project Status Tab—provides an overview of the project settings and at-a-glance summary of synthesis messages and reports. • Implementation Directory—lists the names and types of the result files, and the dates they were last modified. • Process View—gives you instant visibility to the synthesis and place-and-route job flows. <p>See The Project Results View, on page 34 for more information.</p>

The Project view has the following main parts:

Project View Interface	Description
Status	Displays the current status of the synthesis job that is running. Clicking in this area displays additional information about the current job (see Job Status Command, on page 240).
Buttons and options	Allow immediate access to some of the more common commands. See Buttons and Options, on page 92 for details.
Project Management view	Lists the projects and implementations, and their associated HDL source files and constraint files. See Projects and Implementations, on page 22 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 309](#)).

Project Management View



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

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

- Project Files Tab
- Design Hierarchy Tab

However, the Hierarchical Project Management flow is not supported for Microsemi designs.

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

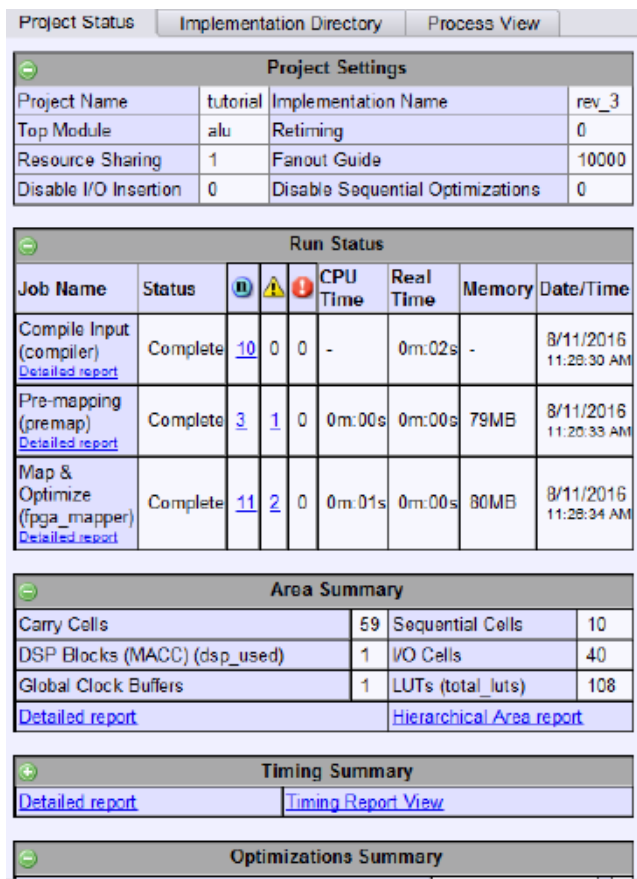
The Project Status view provides an overview of the project settings and at-a-glance summary of synthesis messages and reports such as an area or optimization summary for the active implementation. You can track the status and settings for your design and easily navigate to reports and messages in the 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 187](#).

You can expand or collapse each section of the Project Status view by clicking



The screenshot shows the 'Project Status' tab with three sub-sections: 'Project Settings', 'Run Status', and 'Area Summary'. Each section has a minus icon (-) in its top-left corner, indicating it is currently expanded. Below 'Run Status' are 'Timing Summary' and 'Optimizations Summary', both with plus icons (+) in their top-left corners, indicating they are collapsed. The 'Project Settings' table lists project details like name, module, and resource sharing. The 'Run Status' table shows the progress of various steps like 'Compile Input', 'Pre-mapping', and 'Map & Optimize'. The 'Area Summary' table provides resource usage statistics like 'Carry Cells', 'DSP Blocks', and 'Global Clock Buffers'.

Project Settings			
Project Name	tutorial	Implementation Name	rev_3
Top Module	alu	Retiming	0
Resource Sharing	1	Fanout Guide	10000
Disable I/O Insertion	0	Disable Sequential Optimizations	0

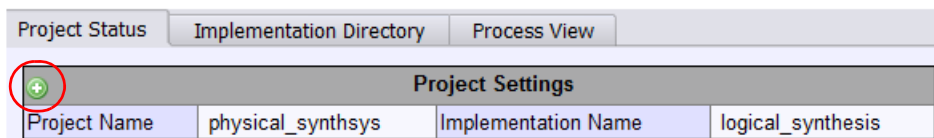
Run Status								
Job Name	Status	0	1	2	CPU Time	Real Time	Memory	Date/Time
Compile Input (compiler)	Complete	10	0	0	-	0m:02s	-	8/11/2016 11:28:30 AM
Pre-mapping (premap)	Complete	3	1	0	0m:00s	0m:00s	79MB	8/11/2016 11:20:33 AM
Map & Optimize (fpga_mapper)	Complete	11	2	0	0m:01s	0m:00s	80MB	8/11/2016 11:28:34 AM

Area Summary			
Carry Cells	59	Sequential Cells	10
DSP Blocks (MACC) (dsp_used)	1	I/O Cells	40
Global Clock Buffers	1	LUTs (total_luts)	108
Detailed report		Hierarchical Area report	

Timing Summary	
Detailed report	Timing Report View

Optimizations Summary	
-----------------------	--

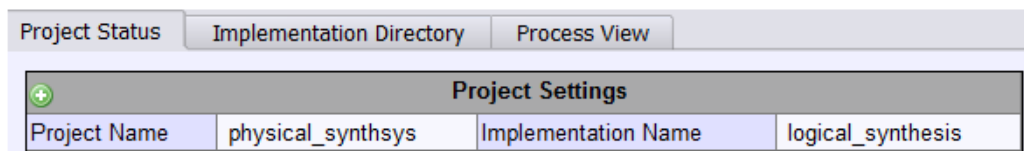
on the + or - icon in the upper left-corner of each section.



This close-up shows the 'Project Settings' section. A red circle highlights the plus icon (+) in the top-left corner of the section header, which is used to expand the section. The table below the header shows 'Project Name' as 'physical_synthesis' and 'Implementation Name' as 'logical_synthesis'.

Project Settings	
Project Name	physical_synthesis
Implementation Name	logical_synthesis

Project Settings



Project Settings			
Project Name	physical_synthesis	Implementation Name	logical_synthesis

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 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.

Run Status							
Job Name	Status				CPU Time	Real Time	Memory
Compile Input Detailed report	Complete	22	0	0	-	0m:02s	-
Premap Detailed report	Complete	4	1	0	0m:00s	0m:00s	78MB
Map & Optimize Detailed report	Complete	15	1	0	0m:16s	0m:17s	102MB

Report: tutorial (rev_3)

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

Copyright (C) 1991-2013, Synopsys, Inc. This software
Product Version I-2013.03 beta
Mapper Startup Complete (Real Time elapsed 0h:00m:0
Linked File: eight_bit_uc_sckk_xvy
Printing clock summary report in "C:\sw\tutorial\z
@N:MF245 : | Running in 32-bit mode.
@N:MF566 : | Clock conversion enabled
Design Input Complete (Real Time elapsed 0h:00m:00s
Mapper Initialization Complete (Real Time elapsed 0
Start loading timing files (Real Time elapsed 0h:00
routetable is 1.6,1.5999994618778257,1.971713156657

- 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
	MF206	Auto Constraint mode is enabled	-	eight_bit_uc_s...	13:29:46...	Pre-mapping Report
	MF600	Clock conversion enabled	-	eight_bit_uc_s...	16:19:24...	Pre-mapping Report
	MF419	Found address in user work space, eight_bit_uc_s...	altv(45)	eight_bit_uc_s...	13:29:44...	Pre-mapping Report
		Found counter in user work space, eight_bit_uc_s...	bcw(33)	eight_bit_uc_s...	13:29:43...	Pre-mapping Report

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

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

Reports

The mapper summary table generates various reports such as an Area Summary, Compile Point Summary, Optimization Summary, and 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	64	60	0	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](#)

Report: tutorial (rev_3)

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

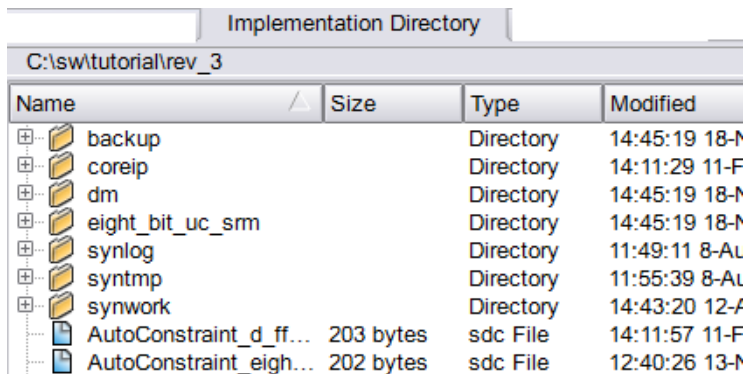
Resource Usage Report for eight_bit_uc

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

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.

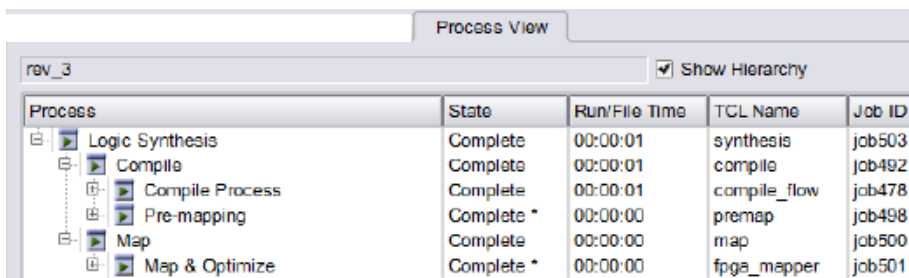


Implementation Directory			
C:\sw\tutorial\rev_3			
Name	Size	Type	Modified
backup		Directory	14:45:19 18-1
coreip		Directory	14:11:29 11-F
dm		Directory	14:45:19 18-1
eight_bit_uc_srm		Directory	14:45:19 18-1
synlog		Directory	11:49:11 8-Al
syntmp		Directory	11:55:39 8-Al
synwork		Directory	14:43:20 12-7
AutoConstraint_d_ff...	203 bytes	sdc File	14:11:57 11-F
AutoConstraint_eigh...	202 bytes	sdc File	12:40:26 13-1

Process View

As process flow jobs become more complex, the benefits of exposing the underlying job flow is extremely valuable. The Process View gives you this visibility to track the design progress for the synthesis and place-and-route job flows.

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



Process	State	Run/File Time	TCL Name	Job ID
Logic Synthesis	Complete	00:00:01	synthesis	job503
Compile	Complete	00:00:01	compile	job492
Compile Process	Complete	00:00:01	compile_flow	job478
Pre-mapping	Complete *	00:00:00	premap	job498
Map	Complete	00:00:00	map	job500
Map & Optimize	Complete *	00:00:00	fpga_mapper	job501

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 44](#)
- [Watch Window, on page 44](#)
- [Tcl Script and Messages Windows, on page 47](#)
- [Tcl Script Window, on page 48](#)
- [Message Viewer, on page 48](#)
- [Output Windows \(Tcl Script and Watch Windows\), on page 52](#)
- [RTL View, on page 53](#)
- [Technology View, on page 54](#)
- [Hierarchy Browser, on page 56](#)
- [FSM Viewer Window, on page 58](#)
- [Text Editor View, on page 60](#)
- [Context Help Editor Window, on page 63](#)
- [Interactive Attribute Examples, on page 64](#)
- [Search SolvNet, on page 66](#)

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

Project view	The Project View, on page 30
SCOPE	SCOPE Tabs, on page 153

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.

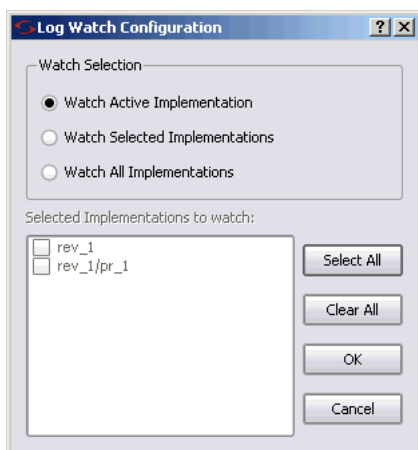
Watch Window

The Watch window displays selected information from the log file (see [Log File, on page 254](#)) as a spreadsheet of parameters that you select to monitor. The values are updated when synthesis finishes.

Watch Window Display

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

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



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

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

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

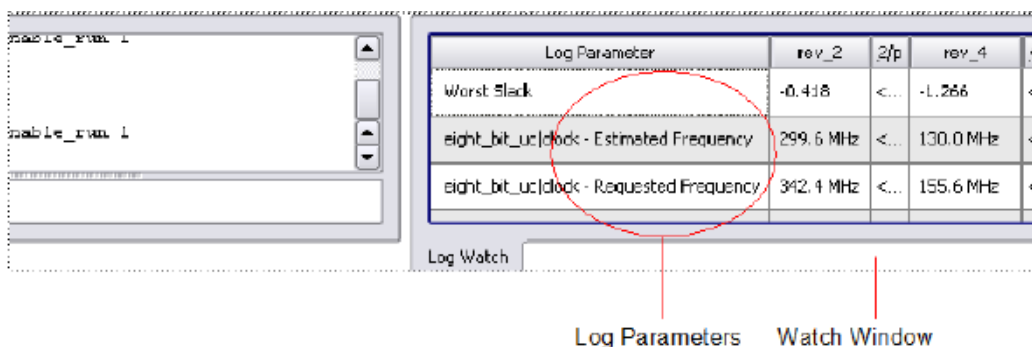
Command	Description
Allow Docking	A toggle: when enabled, the window can be docked.
Hide	Hides the window; use View ->Watch Window to show it again.
Float in Main Window	A toggle: when enabled, the window is floated (undocked).

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

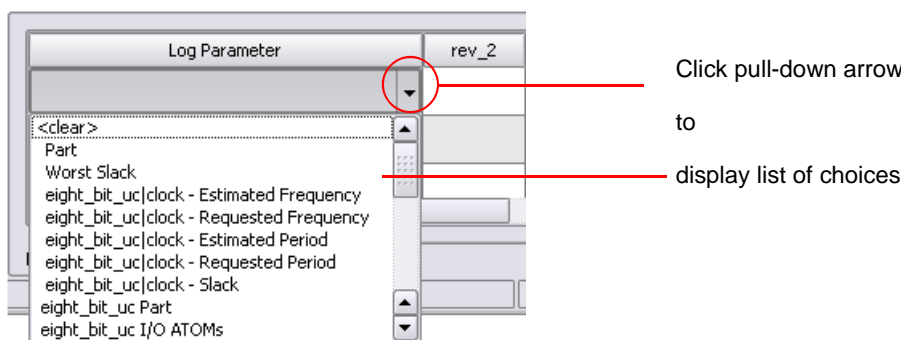
Using the Watch Window

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

To choose log parameters from a pull-down menu, click in the Log Parameter



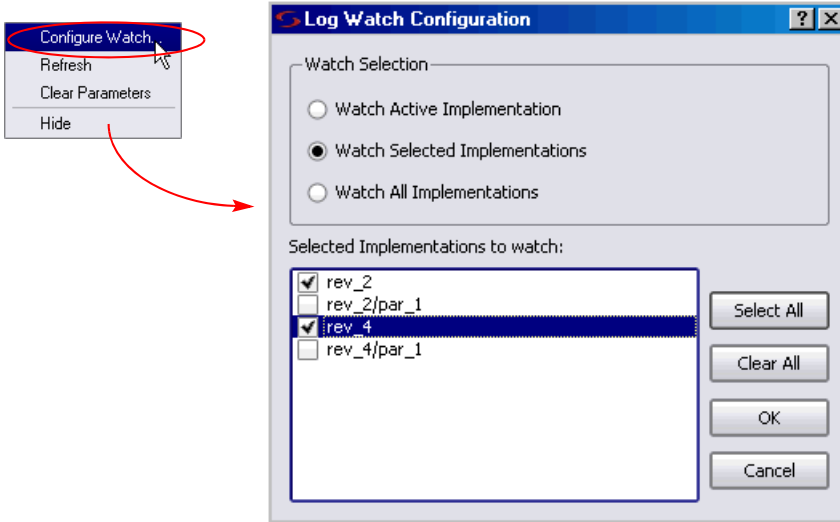
section of the window. Click the pull-down arrow that appears to display the parameter list choices:



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

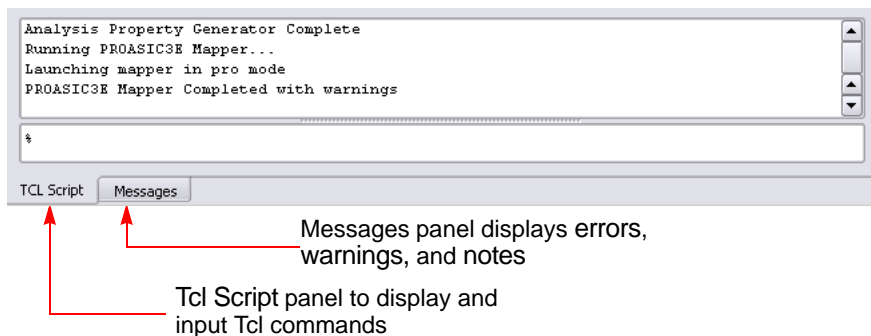
Log Parameter	rev_2	rev_4
Worst Slack	-0.418	-1.266
eight_bit_uc clock - Estimated Frequency	299.6 MHz	130.0 MHz
eight_bit_uc clock - Requested Frequency	342.4 MHz	155.6 MHz

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



Tcl Script and Messages Windows

The Tcl window has tabs for the Tcl Script and Messages windows. By default, the Tcl windows are located below the Project Tree view in the lower left corner of the main application window.



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

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

For more information about the Tcl windows, see [Tcl Script Window, on page 48](#) and [Message Viewer, on page 48](#).

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.

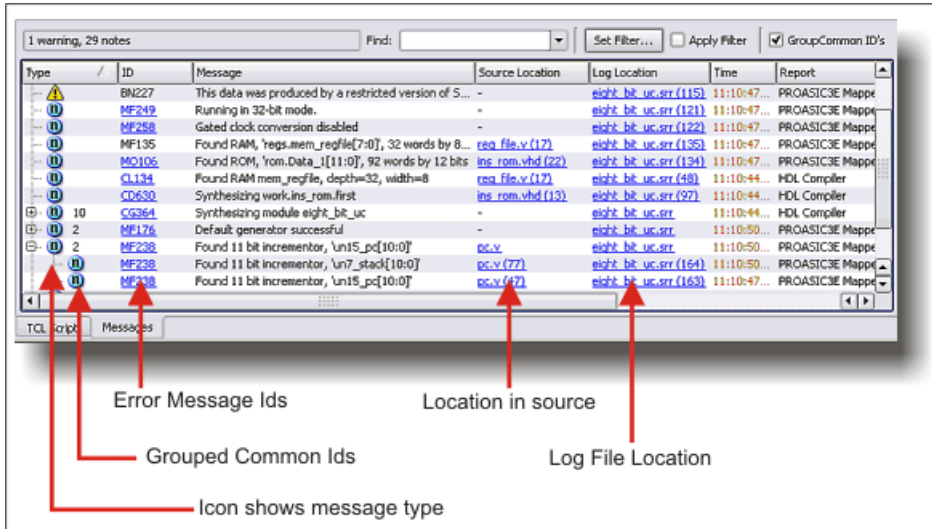
The Tcl script window also displays each command executed in the course of running the synthesis tool, regardless of whether it was initiated from a menu, button, or keyboard shortcut. Right-clicking inside the Tcl window displays a popup menu with the Copy, Paste, Hide, and Help commands.

See also

- [Synthesis Commands, on page 89](#), for information about the Tcl synthesis commands.
- [Generating a Job Script, on page 523](#) in the *User Guide*.

Message Viewer

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







Interactive tasks in the Messages panel include:

- Drag the pane divider with the mouse to change the relative column size.
- Click 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.

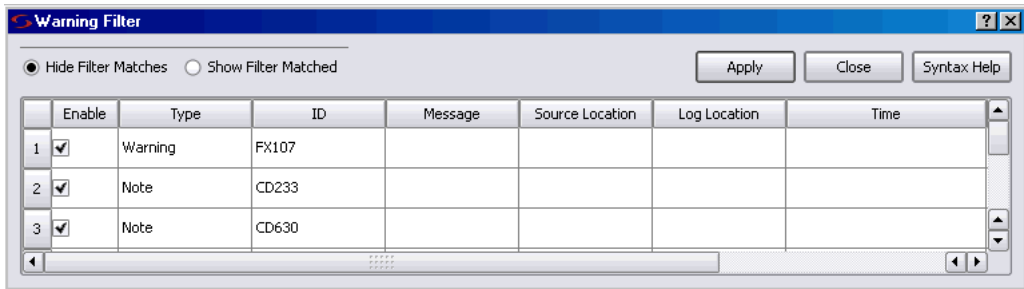
The following table describes the contents of the Messages panel. You can sort the messages by clicking the column headers. For further sorting, use Find and Filter. For details about using this window, see [Checking Results in the Message Viewer, on page 197](#) in the *User Guide*.

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 51 .
Apply Filter	Enable/disable the last saved filter.

Item	Description
Group Common ID's	<p>Enable/disable grouping of repeated messages. Groups are indicated by a number next to the type icon. There are two types of groups:</p> <ul style="list-style-type: none"> • The same warning or note ID appears in multiple source files indicated by a dash in the source files column. • Multiple warnings or notes in the same line of source code indicated by a bracketed number.
Type	<p>The icons indicate the type of message:</p> <p> Error</p> <p> Warning</p> <p> Note</p> <p> Advisory</p> <p>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.</p>
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	<p>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.</p> <p>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.</p>
Report	Indicates which section of the Log File report the error appears, for example Compiler or Mapper.

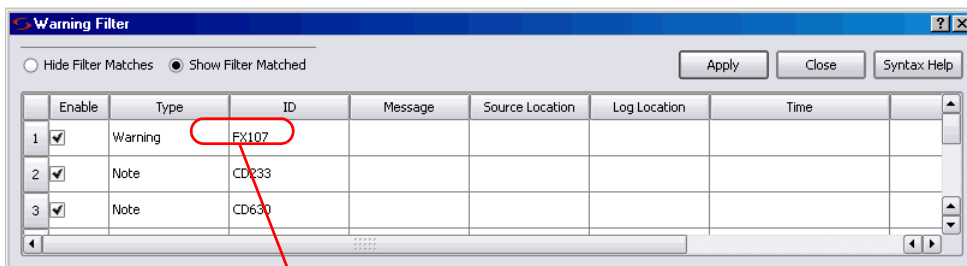
Messages Filter

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

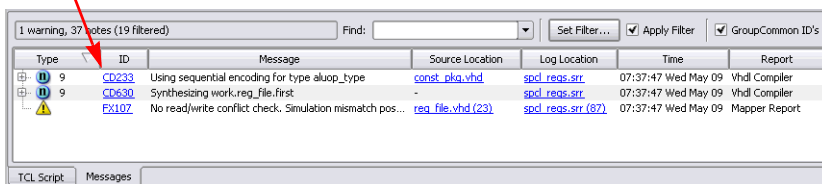


Item	Description
Hide Filter Matches	Hides matched criteria in the Messages Panel.
Show Filter Matches	Shows matched criteria in the Messages Panel.
Syntax Help	Gives quick syntax descriptions.
Apply	Applies the filter criteria to the Messages Panel report, without closing the window.
Type, ID, Message,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

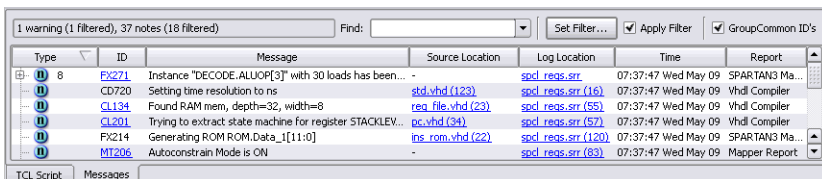
The following is a filtering example.



Show Filter
Matches



Hide Filter
Matches




Output Windows (Tcl Script and Watch Windows)

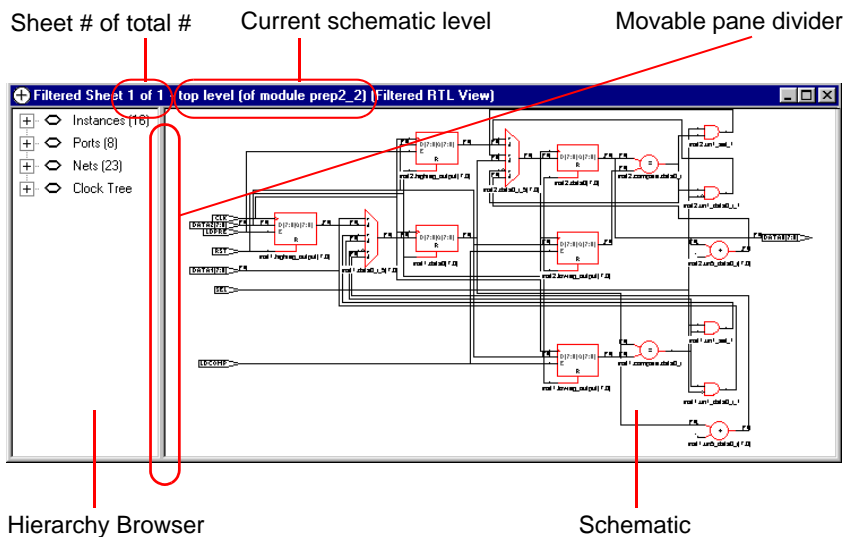
The Output windows are the Tcl Script and Log Watch windows. To display or hide them, use View->Output Windows from the main menu. Refer to [Watch Window, on page 44](#) and [Tcl Script and Messages Windows, on page 47](#) for more information.

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 54](#).

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 56](#). Your design is drawn as a set of schematics. The schematic for a design module (or the top level) consists of one or more sheets, only one of which is visible in a given view at any time. The title bar of the window indicates the current hierarchical schematic level, the current sheet, and the total number of sheets for that level.



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


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

For information about ... See ...

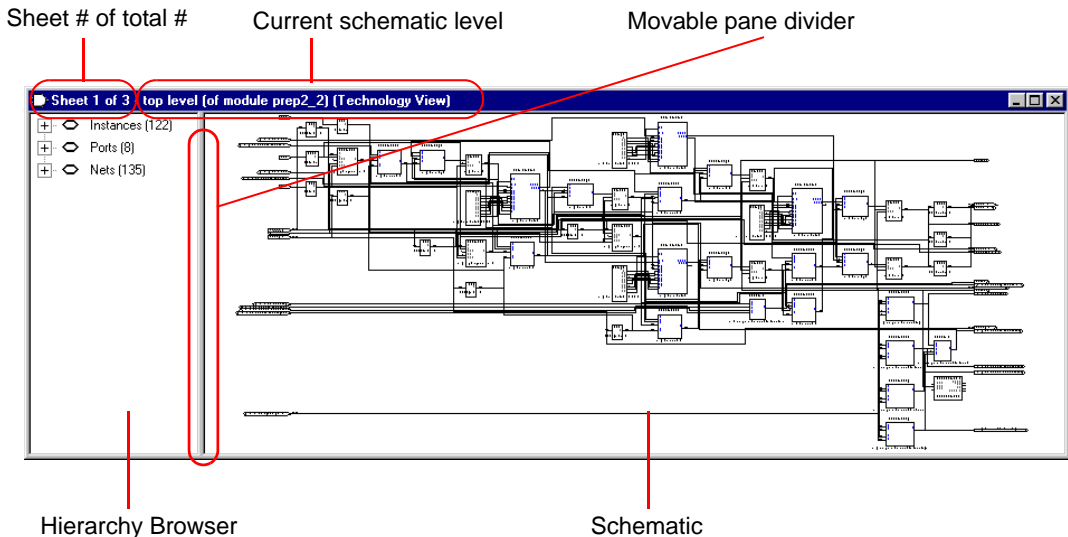
Hierarchy Browser	Hierarchy Browser, on page 56
Procedures for RTL view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Standard Schematic, on page 255 of the <i>User Guide</i> .
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool, on page 95
Commands for RTL view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands, on page 97 HDL Analyst Menu, on page 293
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands, on page 182
History commands: Back and Forward	View Menu: RTL and Technology Views Commands, on page 182
Search command	Find Command (HDL Analyst), on page 173

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 53](#).

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 56](#). Your design is drawn as a set of schematics at different design levels. The schematic for a design module (or the top level) consists of one or more sheets, only one of which is visible in a given view at any time. The title bar of the window indicates the current schematic level, the current sheet, and the total number of sheets for that level.



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

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

For information about ... See ...

Hierarchy Browser	Hierarchy Browser, on page 56
Procedures for Technology view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Standard Schematic, on page 255 of the User Guide
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool, on page 95
Commands for Technology view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands, on page 97 HDL Analyst Menu, on page 293
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands, on page 182
History commands: Back and Forward	View Menu: RTL and Technology Views Commands, on page 182
Search command	Find Command (HDL Analyst), on page 173

Hierarchy Browser

The Hierarchy Browser is the left pane in the RTL and Technology views. (See [RTL View, on page 53](#) and [Technology View, on page 54](#).) 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 57](#) 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 118](#) for more information about using the Hierarchy Browser for navigation and crossprobing.

Hierarchy Browser Symbols

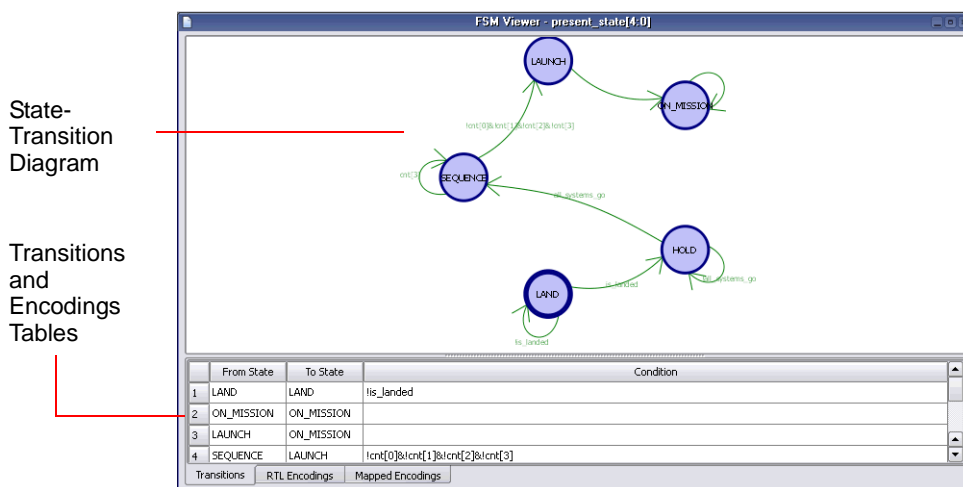
Common symbols used in Hierarchy Browsers are listed in the following table.

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


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

FSM Viewer Window

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



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

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

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

The following table describes FSM Viewer operations.

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

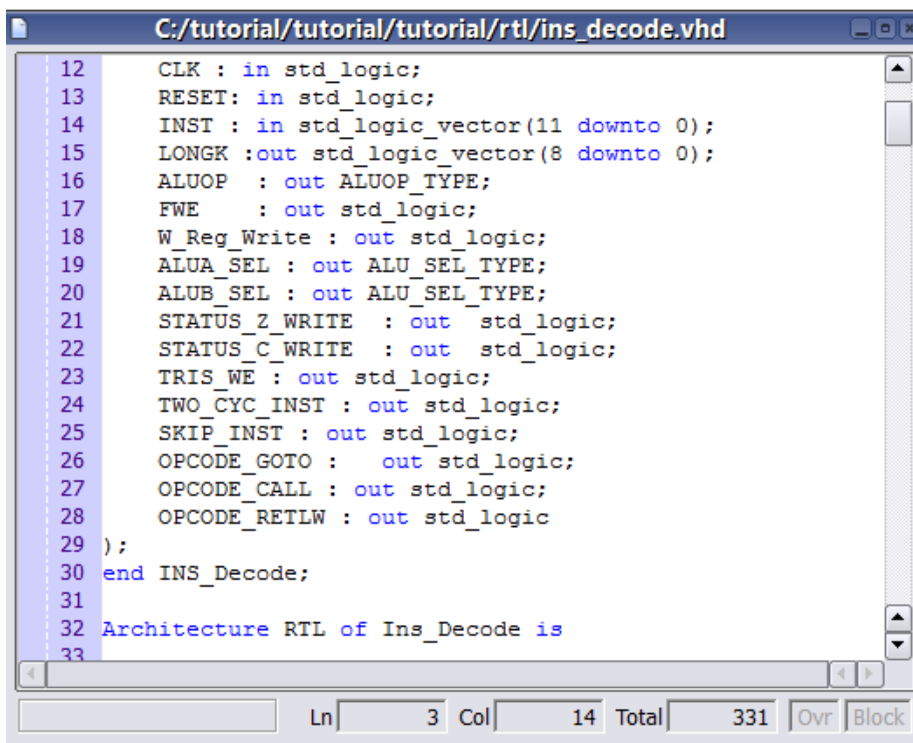
See also:

- [Pushing and Popping Hierarchical Levels, on page 115](#), for information on the operation of pushing into a state machine.
- [FSM Viewer Toolbar, on page 81](#), for information on the FSM icons.

- See [Using the FSM Viewer, on page 317](#) 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 34](#) in the *User Guide*.




Opening the Text Editor

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

- Double-click a source code file (v or vhd) in the Project view.
- Choose File ->Open. In the dialog box displayed, double-click a file to open it.

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



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


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

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

Text Editor Features

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

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

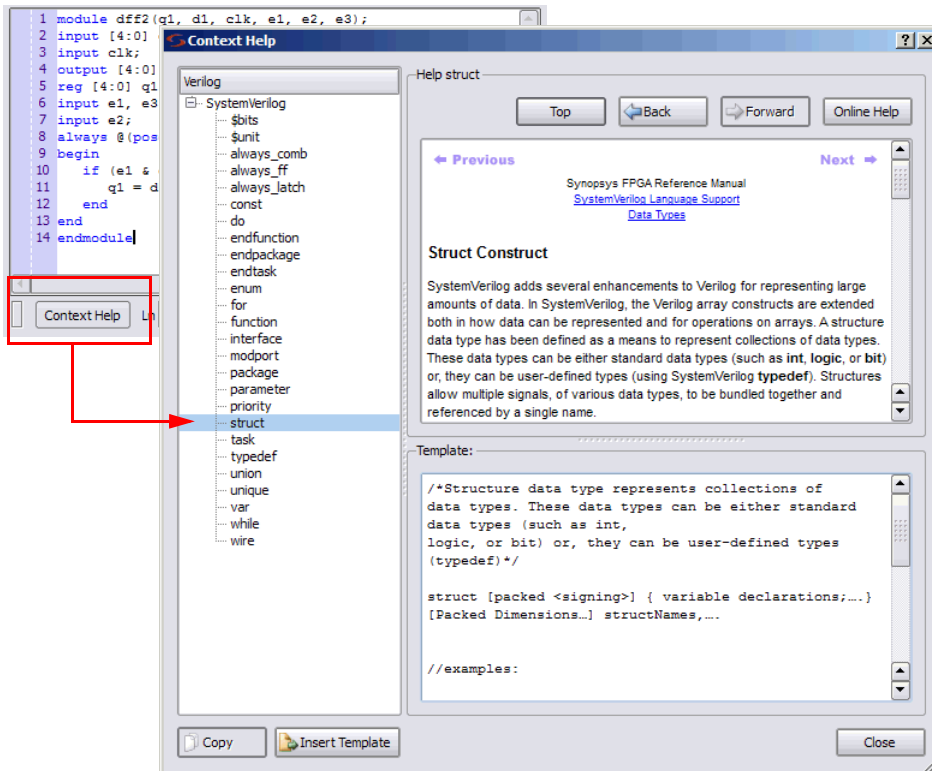
Feature	Description
Deleting all bookmarks	Choose Edit ->Delete all Bookmarks, type Ctrl-Shift-F2, or click the Clear All Bookmarks icon () on the Edit toolbar.
Editing columns	Press and hold Alt, then drag the mouse down a column of text to select it.
Commenting out code	Choose Edit ->Advanced ->Comment Code. The rest of the current line is commented out: the appropriate comment prefix is inserted at the current text cursor position.
Checking syntax	Use Run ->Syntax Check to highlight syntax errors, such as incorrect keywords and punctuation, in source code. If the active window shows an HDL file, then only that file is checked. Otherwise, the entire project is checked.
Checking synthesis	Use Run ->Synthesis Check to highlight hardware-related errors in source code, like incorrectly coded flip-flops. If the active window shows an HDL file, then only that file is checked. Otherwise, the entire project is checked.

See also:

- [Editor Options Command, on page 314](#), for information on setting Text Editor preferences.
- [File Menu, on page 162](#), for information on printing setup operations.
- [Edit Menu Commands for the Text Editor, on page 168](#), for information on Text Editor editing commands.
- [Text Editor Popup Menu, on page 337](#), for information on the Text Editor popup menu.
- [Text Editor Toolbar, on page 80](#), for information on bookmark icons of the Edit toolbar.
- [Keyboard Shortcuts, on page 84](#), for information on keyboard shortcuts that can be used in the Text Editor.

Context Help Editor Window

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



When you select a construct in the left-side of the window, the online help description for the construct is displayed. If the selected construct has this feature enabled, the online help topic is displayed on the top of the window and a generic code or command template for that construct is displayed at the bottom. The Insert Template button is also enabled. When you click the Insert Template button, the code or command shown in the template window is inserted into your file at the location of the cursor. This allows you to easily insert the code or constraint command and modify it for the design that you

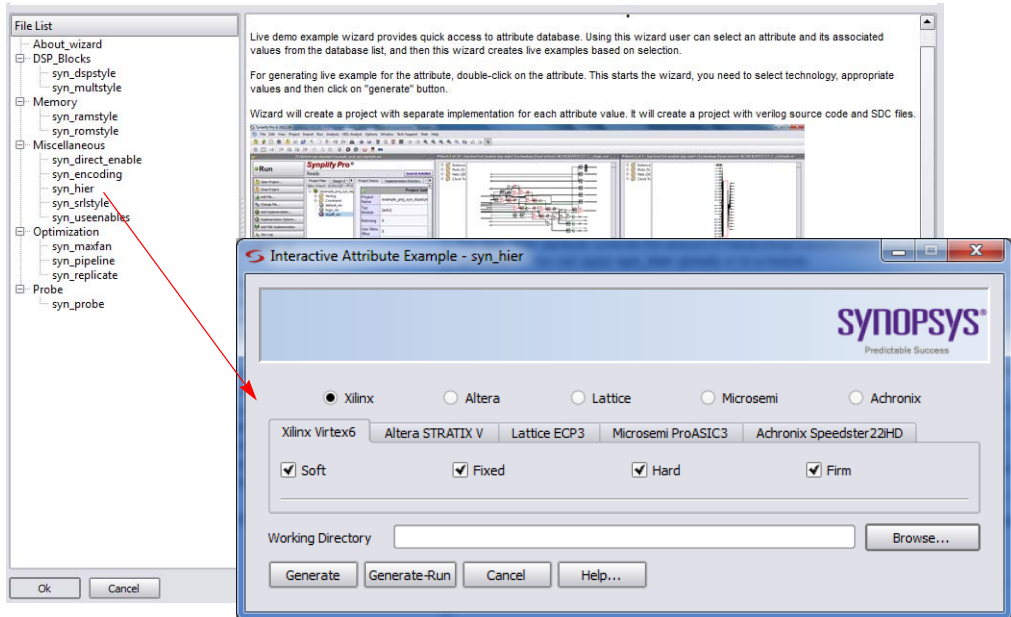
are going to synthesize. If you want to copy only parts of the template, select the code or constraint command you want to insert and click Copy. You can then paste it into your file.

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

Interactive Attribute Examples

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

1. Click Help. Then click Interactive Attribute Examples and the Launch Interactive Attributes Wizard links.



Double-click attribute to bring up the Interactive Attribute wizard

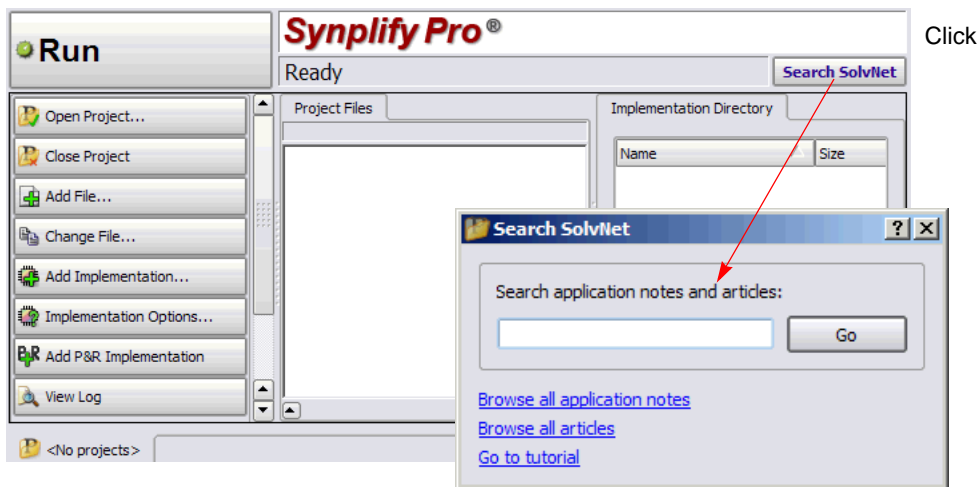
2. Double-click an attribute to start the wizard.
3. Specify the Working Directory location to write your project.
4. Click Generate to generate a project for your attribute.

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.

Search SolvNet

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



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

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

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

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

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 58](#).

See also:

- [Log File, on page 254](#), for information on the log file.
- [RTL and Technology Views Popup Menus, on page 358](#), 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 404](#) 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 <code>syn_state_machine</code> directive to 1 (0), or the VHDL <code>syn_state_machine</code> directive to true (false), for that instance of the state register.

FSM Explorer

The FSM Explorer automatically explores different encoding styles for state machines and picks the style best suited to your design. The FSM explorer runs the FSM viewer to identify the finite state machines in a design, then analyzes the FSMs to select the optimum encoding style for each.

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

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

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

For more information about using the FSM Explorer, see [Running the FSM Explorer, on page 409](#) in the *User Guide*.

Using the Mouse

The mouse button operations in Synopsys FPGA products are standard, see [Mouse Operation Terminology](#) for a summary of supported functions. The Synopsys tool also provides support for:

- [Using Mouse Strokes, on page 70](#)
- [Using the Mouse Buttons, on page 72](#)
- [Using the Mouse Wheel, on page 74](#)

Mouse Operation Terminology

The following terminology is used to refer to mouse operations:

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

Using Mouse Strokes

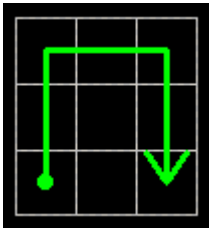
Mouse strokes are used to quickly perform simple repetitive commands. Mouse strokes are drawn by pressing and holding the right mouse button as you draw the pattern. The stroke must be at least 16 pixels in width or height to be recognized. You will see a green mouse trail as you draw the stroke (the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

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

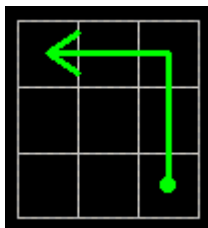
Operating System	Mouse Button + Keyboard Key(s)
Microsoft® Windows®	left mouse button with Alt keyboard key
Linux with UNIX-style keyboard	left mouse button with Meta or diamond keyboard key
Linux with PC-style keyboard	left mouse button with both Ctrl and Alt keyboard keys

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

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



Redo Last Operation

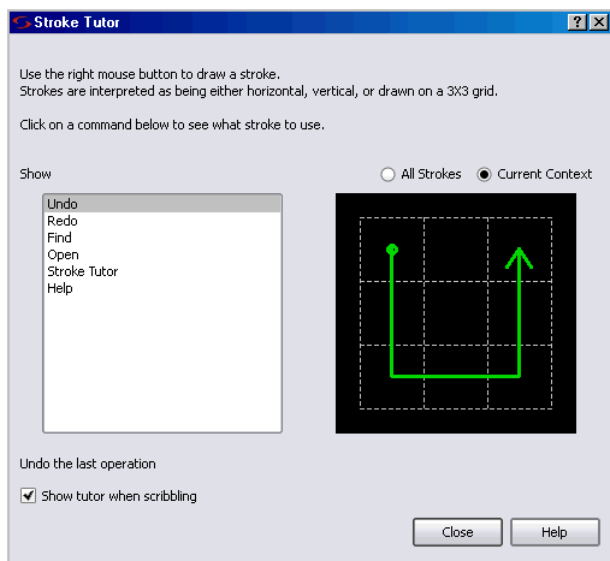


Back to Previous View

The Mouse Stroke Tutor

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

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



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

Mouse strokes are context sensitive. When viewing the Stroke Tutor, you can choose All Strokes or Current Context to view just the strokes that apply to the context of where you invoked the tutor. For example, if you draw the "?" stroke in an 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 53](#), and [Technology View, on page 54](#), 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.

User Interface Preferences

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

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

Managing Views

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

1. Enable the option View ->Workbook Mode.

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



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

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

3. To activate the next view and bring it to the front, type Ctrl-F6. Repeating this keyboard shortcut cycles through all open views. If the next view was minimized it remains minimized, but it is brought to the front so that you can restore it.
4. To close a view, type Ctrl-F4 in the view, or choose File ->Close.
5. You can rearrange open windows using the Window menu: you can cascade them (stack them, slightly offset), or tile them horizontally or vertically.

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.

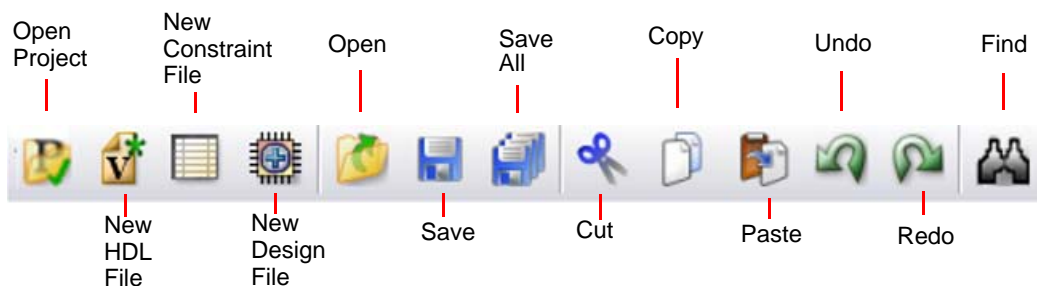
You can enable or disable the display of individual toolbars – see [Toolbar Command, on page 184](#).

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:

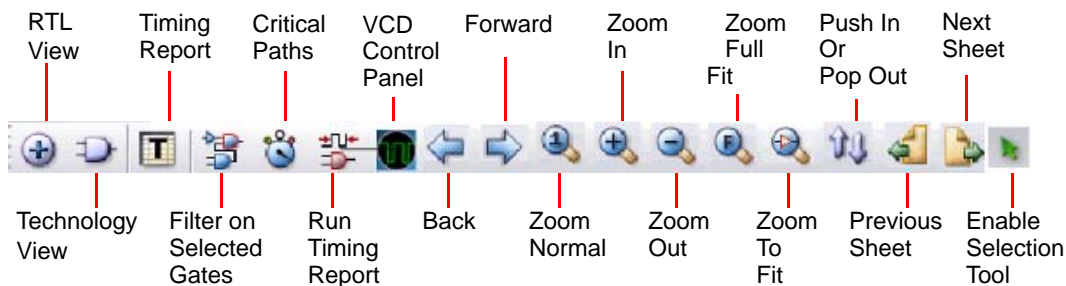
- [File Menu, on page 162](#)
- [Edit Menu, on page 167](#)

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 293](#), for more information.

Icon	Description
 RTL View	<p>Opens a new, hierarchical RTL view: a register transfer-level schematic of the compiled design, together with the associated Hierarchy Browser.</p> <p>Same as HDL Analyst ->RTL ->Hierarchical View.</p>
 Technology View	<p>Opens a new, hierarchical Technology view: a technology-level schematic of the mapped (synthesized) design, together with the associated Hierarchy Browser.</p> <p>Same as HDL Analyst ->Technology ->Hierarchical View.</p>
 Timing Report View	Not available for Microsemi designs.
 Filter Schematic	<p>Filters your entire design to show only the selected objects. The result is a <i>filtered</i> schematic.</p> <p>Same as HDL Analyst ->Filter Schematic.</p>
 Show Critical Path	<p>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.</p> <p>Available only in a Technology view. Not available in a Timing view.</p> <p>Same as HDL Analyst ->Show Critical Path.</p>
 Timing Analyst	<p>Generates and displays a custom timing report and view. The timing report provides more information than the default report (specific paths or more than five paths) or one that provides timing based on additional analysis constraint files. See Analysis Menu, on page 281.</p> <p>Only available for certain device technologies.</p> <p>Same as Analysis ->Timing Analyst.</p>
 VCD Panel	Not available for Microsemi designs.
 Back	<p>Goes backward in the history of displayed sheets of the current HDL Analyst view.</p> <p>Same as View ->Back.</p>

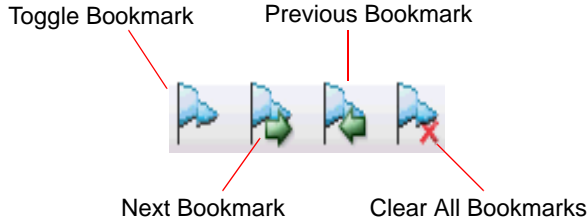
Icon	Description
 Forward	Goes forward in the history of displayed sheets of the current HDL Analyst view. Same as View ->Forward.
 Zoom 100%	Zooms in at a 1:1 ratio and centers the active view where you click. If the view is already normal size, it re-centers the view at the new click location. Same as View ->Normal View. ^a
 Zoom In	Zooms the view in or out. Buttons stay active until deselected. Same as View ->Zoom In or View ->Zoom Out. ^a
 Zoom Out	
 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 76.](#)) The Edit toolbar provides the following icons, by default:

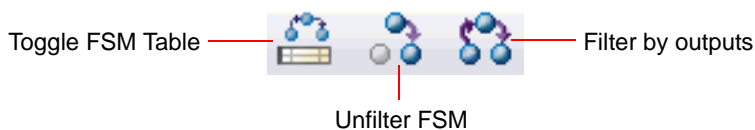


The following table describes the default Edit icons. Each is available in the Text Editor, and each is equivalent to an Edit menu command there – see [Edit Menu Commands for the Text Editor, on page 168](#), for more information.




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

FSM Viewer Toolbar

When you push down into a state machine primitive in an RTL view, the FSM Viewer displays and enables the FSM toolbar. 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 181](#), 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 (<i>project_name_cck.rpt</i>). Same as Run->Constraint Check.
 Launch Identify Instrumentor	Launches the Synopsys Identify Instrumentor product. For more information, see Working with the Identify Tools, on page 552 of the User Guide.

Icon	Description
	Launch Identify Debugger Launches the Synopsys Identify Debugger product. For more information, see Working with the Identify Tools, on page 552 of the User Guide.
	Launch SYNCore Launches the SYNCore IP wizard. This tool helps you build IP blocks such as memory models for your design. For more information, see Launch SYNCore Command, on page 242 .
	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.

Run	
Resynthesize All	
Compile Only	F7
Write Output Netlist Only	
Estimate Area	F9
Compile Physical Hierarchy	Shift+F9
FSM Explorer	F10
Translate Constraints...	
Syntax Check	Shift+F7
Synthesis Check	Shift+F8
 Constraint Check	Shift+F10
Arrange VHDL Files	
 Launch Identify	
 Launch Identify Debugger	
 Launch SYNCORE...	
 Configure and Launch VCS Simulator ...	
Run TCL Script...	
Run All Implementations	
Job Status	
Next Error/Warning	F5
Previous Error/Warning	Shift+F5

The following table describes the keyboard shortcuts.

Keyboard Shortcut	Description
b	<p>In an RTL or Technology view, shows all logic between two or more selected objects (instances, pins, ports). The result is a <i>filtered</i> schematic. Limited to the current schematic.</p> <p>Same as HDL Analyst ->Current Level ->Expand Paths (see HDL Analyst Menu: Filtering and Flattening Commands, on page 296).</p>
Ctrl++ (number pad)	<p>In the FSM Viewer, hides all but the selected state(s), their output transitions, and the destination states of those transitions.</p> <p>Same as View ->Filter ->By output transitions.</p>
Ctrl+- (number pad)	<p>In the FSM Viewer, hides all but the selected state(s), their input transitions, and the origin states of those transitions.</p> <p>Same as View ->Filter ->By input transitions.</p>
Ctrl+* (number pad)	<p>In the FSM Viewer, hides all but the selected state(s), their input and output transitions, and their predecessor and successor states.</p> <p>Same as View ->Filter ->By any transition.</p>
Ctrl-1	<p>In an RTL or Technology view, zooms the active view, when you click, to full (normal) size. Same as View ->Normal View.</p>
Ctrl-a	<p>Centers the window on the design. Same as View ->Pan Center.</p>
Ctrl-b	<p>In an RTL or Technology view, shows all logic between two or more selected objects (instances, pins, ports). The result is a <i>filtered</i> schematic. Operates hierarchically, on lower levels as well as the current schematic.</p> <p>Same as HDL Analyst ->Hierarchical ->Expand Paths (see HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 294).</p>
Ctrl-c	<p>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 173.)</p>
Ctrl-d	<p>In an RTL or Technology view, selects the driver for the selected net. Operates hierarchically, on lower levels as well as the current schematic.</p> <p>Same as HDL Analyst->Hierarchical ->Select Net Driver (see HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 294).</p>

Keyboard Shortcut	Description
Ctrl-e	<p>In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, to the nearest objects (no farther). The result is a <i>filtered</i> schematic. Operates hierarchically, on lower levels as well as the current schematic.</p> <p>Same as HDL Analyst->Hierarchical ->Expand (see HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 294).</p>
Ctrl-Enter (Return)	<p>In the FSM Viewer, hides all but the selected state(s).</p> <p>Same as View->Filter->Selected (see View Menu, on page 181).</p>
Ctrl-f	Finds the selected object. Same as Edit->Find.
Ctrl-F2	<p>Alternately inserts and removes a bookmark to the line that contains the text cursor.</p> <p>Same as Edit->Toggle bookmark (see Edit Menu Commands for the Text Editor, on page 168).</p>
Ctrl-F4	Closes the current window. Same as File ->Close.
Ctrl-F6	Toggles between active windows.
Ctrl-g	<p>In the Text Editor, jumps to the specified line. Same as Edit->Goto (see Edit Menu Commands for the Text Editor, on page 168).</p> <p>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 182).</p>
Ctrl-h	In the Text Editor, replaces text. Same as Edit->Replace (see Edit Menu Commands for the Text Editor, on page 168).
Ctrl-i	<p>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 294).</p>
Ctrl-j	<p>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.</p> <p>Same as HDL Analyst->Hierarchical->Goto Net Driver (see HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 294).</p>

Keyboard Shortcut	Description
Ctrl-l	<p>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.</p> <p>Same as View->Zoom Lock (see View Menu Commands: All Views, on page 181).</p>
Ctrl-m	<p>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 294).</p>
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	<p>In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, until registers, ports, or black boxes are reached. The result is a <i>filtered</i> schematic. Operates hierarchically, on lower levels as well as the current schematic.</p> <p>Same as HDL Analyst->Hierarchical->Expand to Register/Port (see HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 294).</p>
Ctrl-s	In the Project View, saves the file. Same as File ->Save.
Ctrl-t	<p>Toggles display of the Tcl window.</p> <p>Same as View ->Tcl Window (see View Menu, on page 181).</p>
Ctrl-u	<p>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 168).</p> <p>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 183).</p>
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 182). 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 182). 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 168).
Ctrl-Shift-h	In an RTL or Technology view, shows all pins on selected <i>transparent</i> hierarchical (non-primitive) instances. Pins on primitives are always shown. Available only in a filtered schematic. Same as HDL Analyst ->Show All Hier Pins (see HDL Analyst Menu: Analysis Commands, on page 300).
Ctrl-Shift-i	In an RTL or Technology view, selects all instances on the current schematic level (all sheets). This does <i>not</i> select instances on other levels. Same as HDL Analyst->Select All Schematic->Instances (see HDL Analyst Menu, on page 293).
Ctrl-Shift-p	In an RTL or Technology view, selects all ports on the current schematic level (all sheets). This does <i>not</i> select ports on other levels. Same as HDL Analyst->Select All Schematic->Ports (see HDL Analyst Menu, on page 293).
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 168).

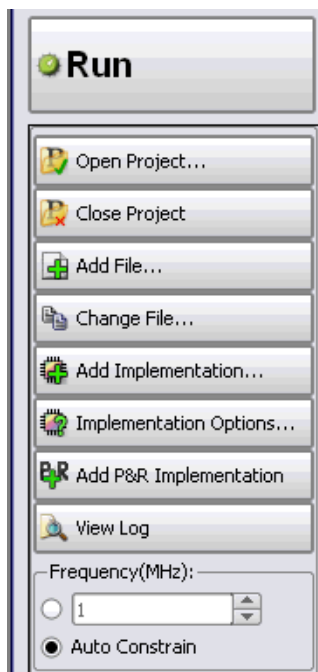
Keyboard Shortcut	Description
d	In an RTL or Technology view, selects the driver for the selected net. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Select Net Driver (see HDL Analyst Menu, on page 293).
Delete (DEL)	Removes the selected files from the project. Same as Project->Remove 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 293).
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 182). 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 168).
F4	In the Project view, adds a file to the project. Same as Project->Add Source File (see Build Project Command, on page 166). 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 182).
F5	Displays the next source file error. Same as Run->Next Error/Warning (see Run Menu, on page 234).
F7	Compiles your design, without mapping it. Same as Run->Compile Only (see Run Menu, on page 234).
F8	Synthesizes (compiles and maps) your design. Same as Run->Synthesize (see Run Menu, on page 234).

Keyboard Shortcut	Description
F10	<p>In the Project view, runs the FSM Explorer to determine optimum encoding styles for finite state machines. Same as Run ->FSM Explorer (see Run Menu, on page 234).</p> <p>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 182).</p>
F11	<p>Toggles zooming in.</p> <p>Same as View->Zoom In (see View Menu: RTL and Technology Views Commands, on page 182).</p>
F12	<p>In an RTL or Technology view, filters your entire design to show only the selected objects.</p> <p>Same as HDL Analyst->Filter Schematic – see HDL Analyst Menu: Filtering and Flattening Commands, on page 296.</p>
i	<p>In an RTL or Technology view, selects instances connected to the selected net. Limited to the current schematic.</p> <p>Same as HDL Analyst->Current Level->Select Net Instances (see HDL Analyst Menu, on page 293).</p>
j	<p>In an RTL or Technology view, displays the unfiltered schematic sheet that contains the net driver for the selected net.</p> <p>Same as HDL Analyst->Current Level->Goto Net Driver (see HDL Analyst Menu, on page 293).</p>
r	<p>In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, until registers, ports, or black boxes are reached. The result is a <i>filtered</i> schematic. Limited to the current schematic.</p> <p>Same as HDL Analyst ->Current Level->Expand to Register/Port (see HDL Analyst Menu, on page 293).</p>
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	<p>Displays the previous source file error.</p> <p>Same as Run->Previous Error/Warning (see Run Menu, on page 234).</p>
Shift-F7	<p>Checks source file syntax.</p> <p>Same as Run->Syntax Check (see Run Menu, on page 234).</p>

Keyboard Shortcut	Description
Shift-F8	Checks synthesis. Same as Run->Synthesis Check (see Run Menu, on page 234).
Shift-F10	Checks the timing constraints in the constraint files in your project and generates a report (<i>project_name_cck.rpt</i>). Same as Run->Constraint Check (see Run Menu, on page 234). 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 303).
Shift-F11	Toggles zooming out. Same as View->Zoom Out (see View Menu, on page 181).
Shift-Left Arrow	Displays the previous sheet of a multiple-sheet schematic.
Shift-Right Arrow	Displays the next sheet of a multiple-sheet schematic.
Shift-s	Dissolves the selected instances, showing their lower-level details. Dissolving an instance one level replaces it, in the current sheet, by what you would see if you pushed into it using the push/pop mode. The rest of the sheet (not selected) remains unchanged. The number of levels dissolved is the Dissolve Levels value in the Schematic Options dialog box. The type (filtered or unfiltered) of the resulting schematic is unchanged from that of the current schematic. However, the effect of the command is different in filtered and unfiltered schematics. Same as HDL Analyst ->Dissolve Instances – see Dissolve Instances, on page 302 .

Buttons and Options

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



The following table describes the Project View buttons and options.

Button/Option	Action
Open Project...	Opens a new or existing project. Same as File->Open Project (see Open Project Command, on page 166).
Close Project	Closes the current project. Same as File->Close Project (see Run Menu, on page 234).
Add File...	Adds a source file to the project. Same as Project->Add Source File (see Build Project Command, on page 166).

Button/Option	Action
Change File...	Replaces one source file with another. Same as Project ->Change File (see Change File Command, on page 194).
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 353 for a description of the dialog box, and Running P&R Automatically after Synthesis, on page 550 in the <i>User Guide</i> for information about using this feature.
View Log	Displays the log file. Same as View ->View Log File (see View Menu, on page 181).
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 342 in the <i>User Guide</i> for detailed information about using this option. You can also set this option on the Constraints panel of the Implementation Options dialog box.
Continue on Error	When enabled for compile-point synthesis, allows the operation to continue on error and synthesize the remaining compile points.
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 67 and Optimizing State Machines, on page 404 in the <i>User Guide</i>).

Button/Option	Action
FSM Explorer	<p>When enabled, the FSM Explorer selects an encoding style for the finite state machines in your design.</p> <p>Same as enabling the FSM Explorer option on the Options panel of the Implementation Options dialog box. For more information, see FSM Explorer, on page 69 and Running the FSM Compiler, on page 405 in the <i>User Guide</i>.</p>
Resource Sharing	<p>When enabled, makes the compiler use resource sharing techniques. This option does not affect resource sharing by the mapper.</p> <p>The option is the same as the Resource Sharing option on the Options panel of the Implementation Options dialog box. See Sharing Resources, on page 402 in the <i>User Guide</i> for usage details.</p>
Retiming	<p>When enabled, improves the timing performance of sequential circuits. The retiming process moves storage devices (flip-flops) across computational elements with no memory (gates/LUTs) to improve the performance of the circuit. This option also adds a retiming report to the log file.</p> <p>Same as enabling the Retiming option on the Options panel of the Implementation Options dialog box. Use the <code>syn_allow_retiming</code> attribute to enable or disable retiming for individual flip-flops. See syn_allow_retiming, on page 46 for syntax details.</p> <p>Note: Pipelining is automatically enabled when retiming is enabled.</p>
Run	<p>Runs synthesis (compilation and mapping).</p> <p>Same as the Run->Synthesize command (see Run Menu, on page 234).</p>

CHAPTER 3

HDL Analyst Tool

The HDL Analyst tool helps you examine your design and synthesis results, and analyze how you can improve design performance and area.

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

- [HDL Analyst Views and Commands, on page 96](#)
- [Schematic Objects and Their Display, on page 98](#)
- [Basic Operations on Schematic Objects, on page 107](#)
- [Multiple-sheet Schematics, on page 112](#)
- [Exploring Design Hierarchy, on page 115](#)
- [Filtering and Flattening Schematics, on page 122](#)
- [Timing Information and Critical Paths, on page 128](#)

For additional information, see the following:

- Descriptions of the HDL Analyst commands in [Chapter 4, User Interface Commands](#):
- [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 53](#) and [Technology View, on page 54](#) 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.

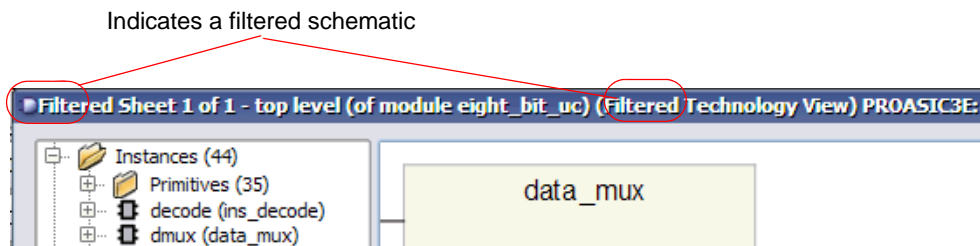
This section gives you information about the following:

- [Filtered and Unfiltered Schematic Views, on page 96](#)
- [Accessing HDL Analyst Commands, on page 97](#)

Filtered and Unfiltered Schematic Views

HDL Analyst views ([RTL View, on page 53](#) and [Technology View, on page 54](#)) consist of schematics that let you analyze your design graphically. The schematics can be filtered or unfiltered. The distinction is important because the kind of view determines how objects are displayed for certain commands.

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



Filtering commands affect only the displayed schematic, not the underlying design. See the following topics:

- For a detailed description of filtering, see [Filtering and Flattening Schematics, on page 122](#).
- For procedures on using filtering, see [Filtering Schematics, on page 302](#) 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 293](#)
- [View Menu, on page 181](#)
- [RTL and Technology Views Popup Menus, on page 358](#)
- [Analyst Toolbar, on page 78](#)

Schematic Objects and Their Display

Schematic objects are the objects that you manipulate in an HDL Analyst schematic: instances, ports, and nets. Instances can be categorized in different ways, depending on the operation: hidden/unhidden, transparent/opaque, or primitive/hierarchical. The following topics describe schematic objects and the display of associated information in more detail:

- [Object Information, on page 98](#)
- [Sheet Connectors, on page 99](#)
- [Primitive and Hierarchical Instances, on page 100](#)
- [Hidden Hierarchical Instances, on page 103](#)
- [Transparent and Opaque Display of Hierarchical Instances, on page 101](#)
- [Schematic Display, on page 103](#)

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

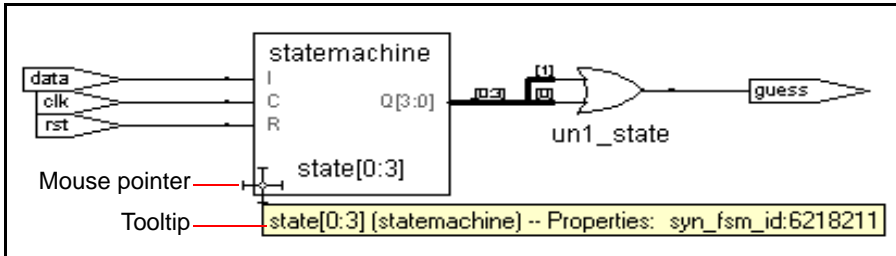
To obtain information about specific objects, you can view object properties with the Properties command from the right-click popup menu, or place the pointer over the object and view the object information displayed. With the latter method, information about the object displays in these two places until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

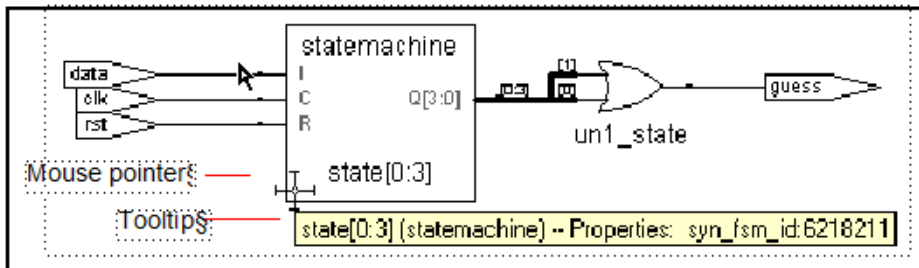
```
Net clock (local net clock) Fanout=4
```

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



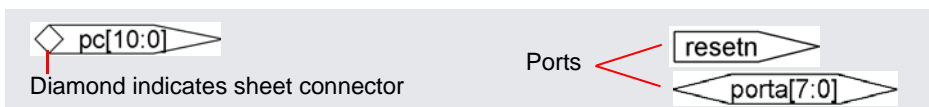
Tooltips option. Do this if you want to reduce clutter.

See also

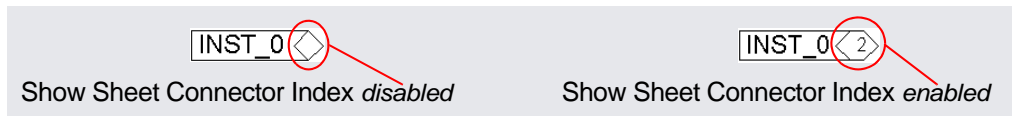
- [Pin and Pin Name Display for Opaque Objects](#), on page 105
- [New HDL Analyst Options Command](#), on page 319

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 326) to control how the schematic is divided into multiple sheets.



If you enable the Show Sheet Connector Index option in the (Options->HDL Analyst Options), the empty diamond becomes a hexagon with a list of the connected sheets. You go to a connecting sheet by right-clicking a sheet connector and choosing the sheet number from the popup menu. The menu has as many sheet numbers as there are sheets connected to the net at that point.



See also

- [Multiple-sheet Schematics, on page 112](#)
- [New HDL Analyst Options Command, on page 319](#)
- [RTL and Technology Views Popup Menus, on page 358](#)

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 57](#)). 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.

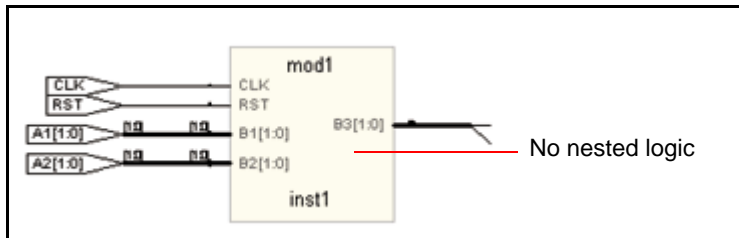
The Hierarchy Browser lists hierarchical instances under *Instances*, and uses this symbol: . In a schematic, the display of hierarchical instances depends on the combination of the following:

- 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 101](#) 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 103](#) for more information.

Transparent and Opaque Display of Hierarchical Instances

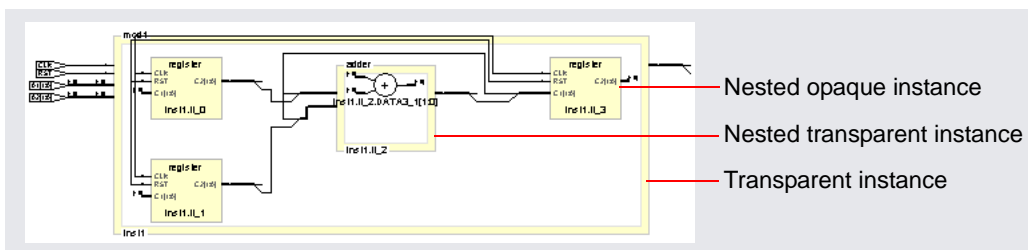
A hierarchical instance can be displayed transparently or opaquely. You cannot directly control the display; certain commands cause instances to be transparent. The distinction between transparent and opaque is important because some commands operate differently on transparent and opaque instances. For 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 120](#) 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.



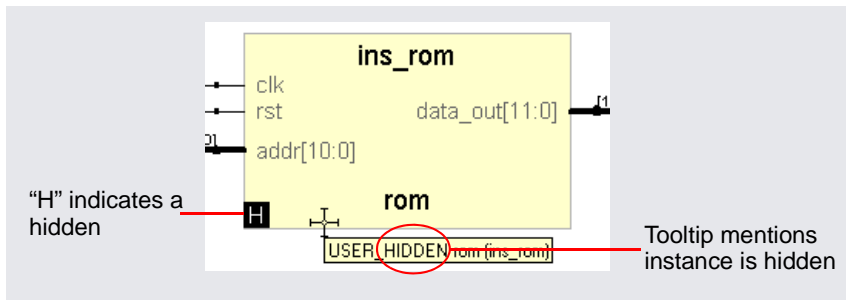
See also

- [Looking Inside Hierarchical Instances, on page 120](#)
- [Multiple Sheets for Transparent Instance Details, on page 114](#)
- [Filtered and Unfiltered Schematic Views, on page 96](#)

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 358](#)). Hiding and unhiding only affects the current HDL Analyst view, and does not affect the Hierarchy Browser. You can hide and unhide instances as needed. The hierarchical logic of a hidden instance is not removed from the design; it is only excluded from certain operations.

The schematics indicate hidden hierarchical instances with a small H in the lower left corner. When the mouse pointer is over a hidden instance, the status bar and the tooltip indicate that the instance is hidden.



Schematic Display

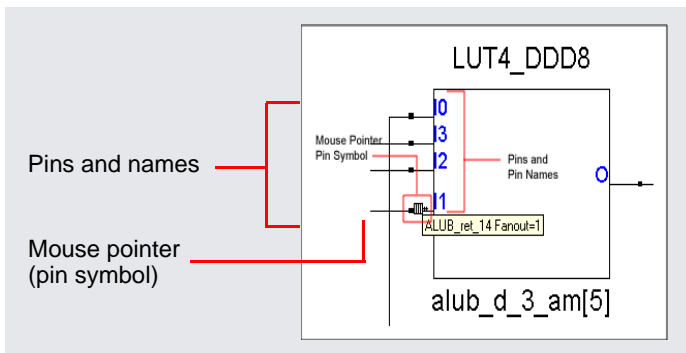
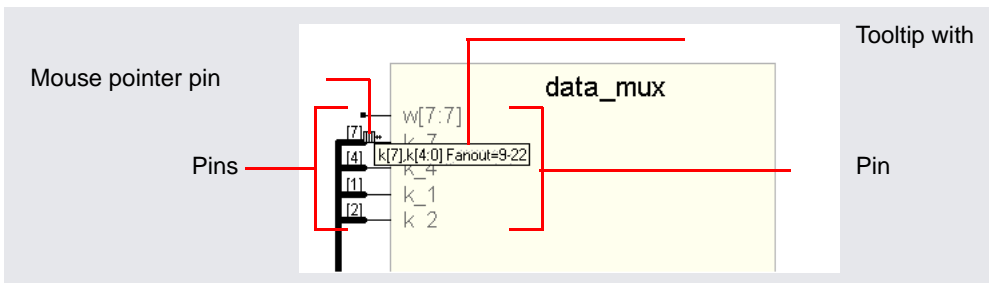
The HDL Analyst Options dialog box controls general properties for all HDL Analyst views, and can determine the display of schematic object information. Setting a display option affects all objects of the given type in all views. Some schematic options only take effect in schematic windows opened after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

The following are some commonly used settings that affect the display of schematic objects. See [New HDL Analyst Options Command, on page 319](#) 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 105 and Pin and Pin Name Display for Transparent Objects, on page 105 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.

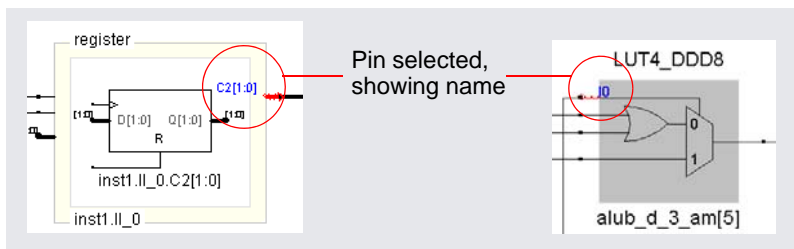
Transparent Hierarchical Instances

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.



See also:

- [New HDL Analyst Options Command, on page 319](#)
- [Controlling the Amount of Logic on a Sheet, on page 112](#)
- [Analyzing Timing in Schematic Views, on page 324 in the *User Guide*](#)

Basic Operations on Schematic Objects

Basic operations on schematic objects include the following:

- [Finding Schematic Objects, on page 107](#)
- [Selecting and Unselecting Schematic Objects, on page 108](#)
- [Crossprobing Objects, on page 109](#)
- [Dragging and Dropping Objects, on page 111](#)

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

- [Filtering and Flattening Schematics, on page 122](#)
- [Timing Information and Critical Paths, on page 128](#)
- [Multiple-sheet Schematics, on page 112](#)
- [Exploring Design Hierarchy, on page 115](#)

Finding Schematic Objects

You can use the following techniques to find objects in the schematic. For step-by-step procedures using these techniques, see [Finding Objects, on page 277](#) 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 277](#) in the *User Guide* for details.

- Edit -> Find command

The Edit -> Find command is described in [Find Command \(HDL Analyst\), on page 173](#). It displays the Object Query dialog box, which lists schematic objects by type (Instances, Symbols, Nets, or Ports) and lets you use wildcards to find objects by name. You can also fine-tune your search by setting a range for the search.

This command selects all found objects, whether or not they are displayed in the current schematic. Although you can search for hidden instances, you cannot find objects that are inside hidden instances at a lower level. Temporarily hiding an instance thus further refines the search range by excluding the internals of a given instance. This can be very useful when working with transparent instances, because the lower-level details appear at the current level, and cannot be excluded by choosing Current Level Only. See [Using Find for Hierarchical and Restricted Searches, on page 279](#) 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 122](#) 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 262](#) in the *User Guide*.

Using the Mouse to Select a Range of Schematic Objects

In a Hierarchy Browser, you can select a *range* of schematic objects by clicking the name of an object at one end of the range, then holding the Shift key while clicking the name of an object at the other end of the range. To use the mouse for selecting and unselecting objects in a schematic, the cross-hairs symbol () must appear as the mouse pointer. If this is not currently the case, right-click the schematic background.

Using Commands to Select Schematic Objects

You can select and deselect schematic objects using the commands in the HDL Analyst menu, or use Edit->Find to find and select objects by name.

The HDL Analyst menu commands that affect selection include the following:

- Expansion commands like Expand, Expand to Register/Port, Expand Paths, and Expand Inwards select the objects that result from the expansion. This means that (except for Expand to Register/Port) you can perform successive expansions and expand the set of objects selected.
- The Select All Schematic and Select All Sheet commands select all instances or ports on the current schematic or sheet, respectively.
- The Select Net Driver and Select Net Instances commands select the appropriate objects according to the hierarchical level you have chosen.
- Deselect All deselects all objects in *all* HDL Analyst views.

See also

- [Finding Schematic Objects, on page 107](#)
- [HDL Analyst Menu, on page 293](#)

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, on page 290](#) in the *User Guide*.

From ...	To ...	Do this ...
Text Editor: log file	Text Editor: HDL source file	Double-click a log file note, error, or warning. The corresponding HDL source code appears in the Text Editor.
Text Editor: HDL code	Analyst view FSM Viewer	<p>The RTL view or Technology view must be open. Select the code in the Text Editor that corresponds to the object(s) you want to crossprobe.</p> <p>The object corresponding to the selected code is automatically selected in the target view, if an HDL source file is in the Text Editor. Otherwise, right-click and choose the Select in Analyst command.</p> <p>To cross-probe from text other than source code, first select Options->HDL Analyst Options and then enable Enhanced Text Crossprobing.</p>
FSM Viewer	Analyst view	<p>The target view must be open. The state machine must be encoded with the onehot style to crossprobe from the transition table.</p> <p>Select a state anywhere in the FSM Viewer (bubble diagram or transition table). The corresponding object is automatically selected in the HDL Analyst view.</p>

From ...	To ...	Do this ...
Analyst view 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 99](#)) mark logic connections from one sheet to the next.

For more information, see

- [Controlling the Amount of Logic on a Sheet, on page 112](#)
- [Navigating Among Schematic Sheets, on page 112](#)
- [Multiple Sheets for Transparent Instance Details, on page 114](#)

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:

- [New HDL Analyst Options Command, on page 319](#)
- [Setting Schematic Preferences, on page 266](#) 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.

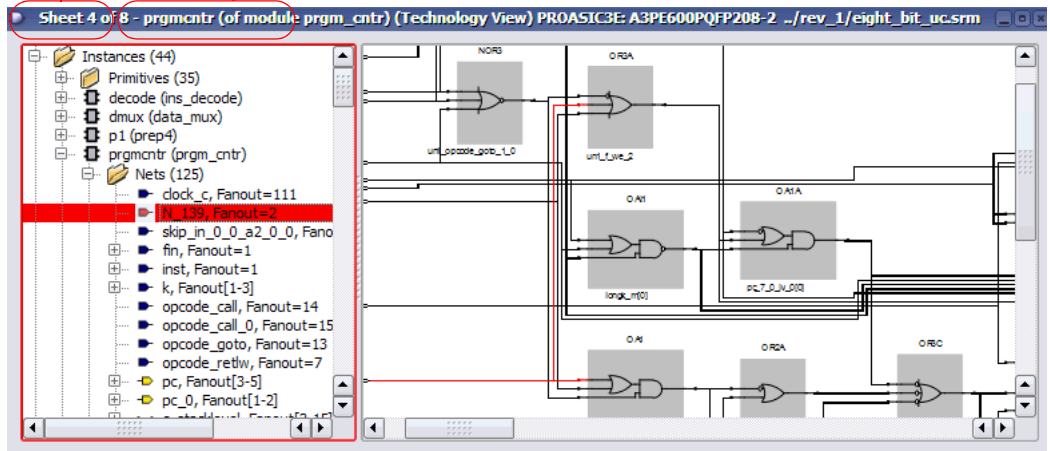
Multisheet Orientation in the Title Bar

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 263 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 122](#)
- [View Menu: RTL and Technology Views Commands, on page 182](#)
- [Pushing and Popping Hierarchical Levels, on page 115](#)

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 112](#).
- 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 112](#)
- [View Menu: RTL and Technology Views Commands, on page 182](#)

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 115](#)
- [Navigating With a Hierarchy Browser, on page 118](#)
- [Looking Inside Hierarchical Instances, on page 120](#)

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 118](#) and [Looking Inside Hierarchical Instances, on page 120](#) 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:

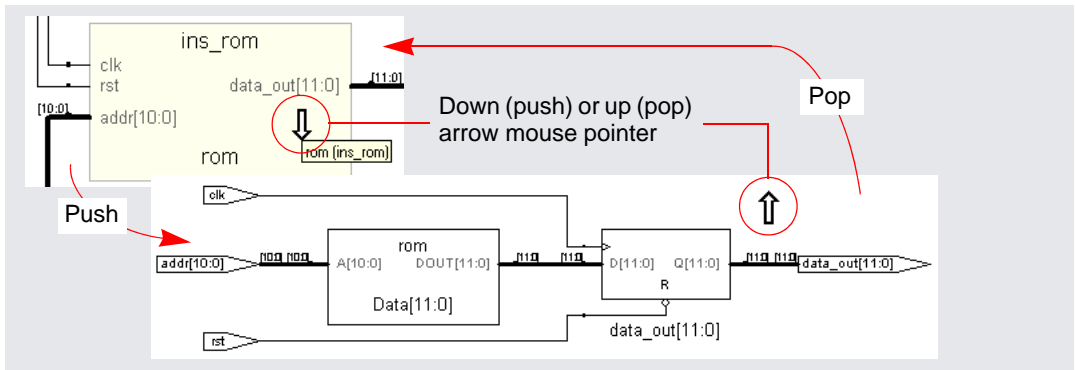
- When you push into an inferred state machine in an RTL view, the FSM Viewer opens, with graphical information about the FSM. See the [FSM Viewer Window, on page 58](#), for more information.
- When you push into an inferred ROM in an RTL view, the Text Editor window opens and displays the ROM data 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 116](#) for details.
- A small H symbol () in the lower left corner indicates a hidden instance, and you cannot push into it.
- The Hierarchy Browser symbols indicates the type of instance and you can use that to determine whether you can push into an object. For example, hierarchical instance (), technology-specific primitive (), logic primitive such as XOR (), or other primitive instance (). The browser symbol does not indicate whether or not an instance is hidden.
- The *status bar* at the bottom of the main synthesis tool window reports information about the object under the pointer, including whether or not it is a hidden instance or a primitive.

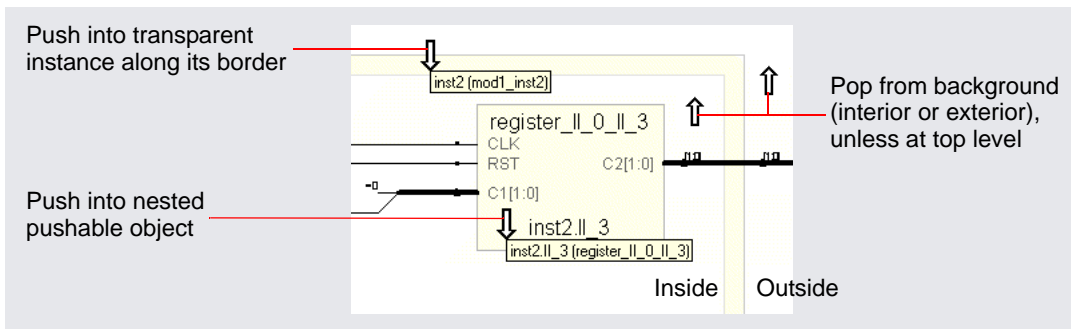
How to Push and Pop Hierarchical Levels

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






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

- To *pop*, place the pointer in an empty area of the schematic background, then click or use the appropriate mouse stroke. The background area inside a transparent instance acts just like the background area outside the instance.
- To *push* into an object, place the mouse pointer over the object and click or use the appropriate mouse stroke. To push into a transparent instance, place the pointer over its pale yellow border, not its hollow (white) interior. Pushing into an object nested inside a transparent hierarchical instance descends to a lower level than pushing into the enclosing transparent instance. In the following figure, pushing into transparent instance `inst2` descends one level; pushing into nested instance `inst2.ll_3` descends two levels.



The following arrow mouse pointers indicate status in Push/Pop mode. For other indicators, see [Pushable Schematic Objects, on page 115](#).

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:

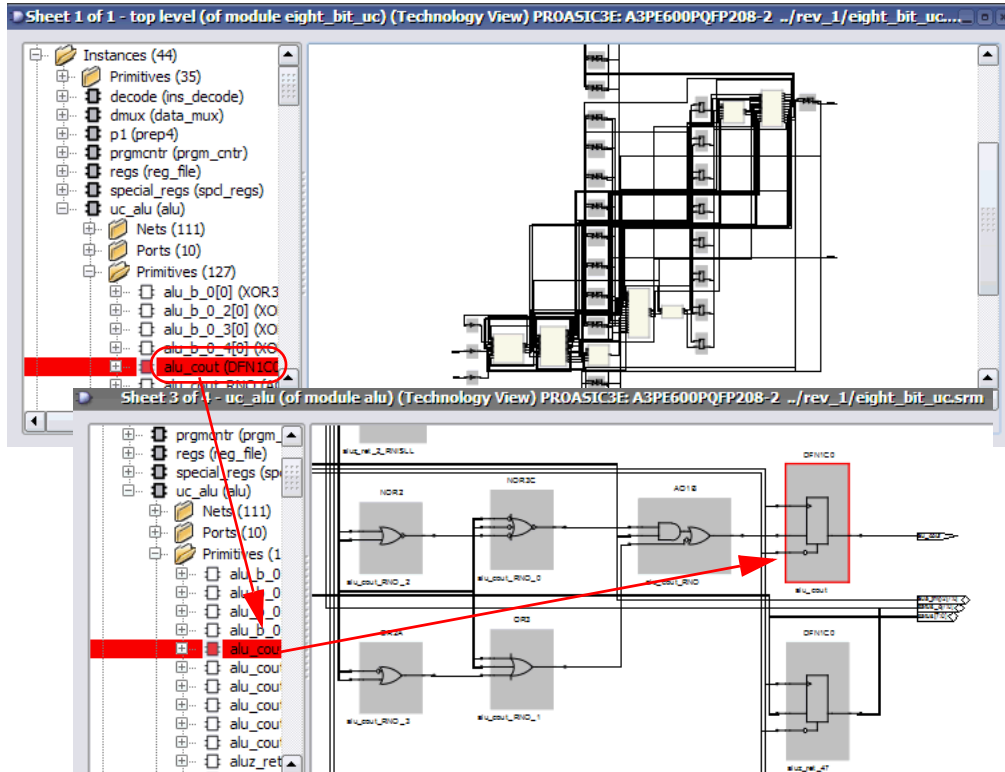
- [Hidden Hierarchical Instances, on page 103](#)
- [Transparent and Opaque Display of Hierarchical Instances, on page 101](#)
- [Using Mouse Strokes, on page 70](#)
- [Navigating With a Hierarchy Browser, on page 118](#)

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 115](#) and [Looking Inside Hierarchical Instances, on page 120](#) for other ways of viewing design hierarchy).

The browser in the RTL view displays the hierarchy specified in the RTL design description. The browser in the Technology view displays the hierarchy of your design after technology mapping.

Selecting an object in the browser displays it in the schematic, because the two are linked. Use the Hierarchy Browser to traverse your hierarchy and select ports, nets, components, and submodules. The browser categorizes the objects, and accompanies each with a symbol that indicates the object type. The following figure shows crossprobing between a schematic and the hierarchy browser.



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

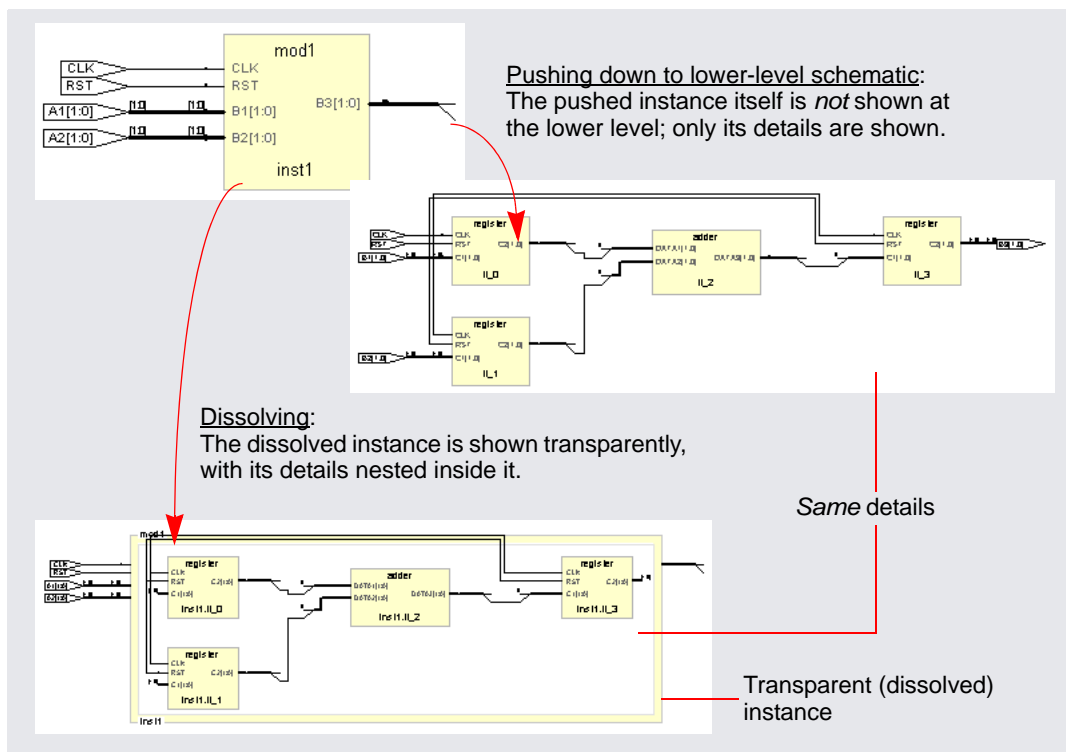
See also:

- [Crossprobing Objects, on page 109](#)
- [Pushing and Popping Hierarchical Levels, on page 115](#)
- [Hierarchy Browser Popup Menu Commands, on page 358](#)

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 118](#) and [Navigating With a Hierarchy Browser, on page 118](#) for other ways of viewing design hierarchy). A transparent instance appears as a hollow box with a pale yellow border. Inside this border are transparent and opaque objects from lower design levels.

Transparent instances provide design context. They show the lower-level logic nested within the transparent instance at the current design level, while pushing shows the same logic a level down. The following figure compares the same lower-level logic viewed in a transparent instance and a push operation:



You cannot control the display of transparent instances directly. However, you can perform the following operations, which result in the display of transparent instances:

- Hierarchically expand an object (using the expansion commands in the HDL Analyst menu).
- Dissolve selected hierarchical instances in a *filtered* schematic (HDL Analyst -> Dissolve Instances).
- Filter a schematic, after selecting multiple objects at more than one level. See [Commands That Result in Filtered Schematics, on page 122](#) 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 115](#)
- [Navigating With a Hierarchy Browser, on page 118](#)
- [HDL Analyst Command, on page 294](#)
- [Transparent and Opaque Display of Hierarchical Instances, on page 101](#)
- [Hidden Hierarchical Instances, on page 103](#)

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 122](#)
- [Combined Filtering Operations, on page 123](#)
- [Returning to The Unfiltered Schematic, on page 123](#)
- [Commands That Flatten Schematics, on page 124](#)
- [Selective Flattening, on page 125](#)
- [Filtering Compared to Flattening, on page 126](#)

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

See also:

- [Filtered and Unfiltered Schematic Views, on page 96](#)
- [HDL Analyst Menu, on page 293](#) and [RTL and Technology Views Popup Menus, on page 358](#)

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 296](#) 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 294](#) 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 294](#) 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 300](#) 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 107](#) 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 358](#)
- [View Menu: RTL and Technology Views Commands, on page 182](#)

Commands That Flatten Schematics

A flattened schematic contains no hierarchical objects. Any command that results in a flattened schematic is a flattening command. This includes the following.

Command	Unfiltered Schematic	Filtered Schematic
Dissolve Instances	Flattens selected instances	--
Flatten Current Schematic (Flatten Schematic)	Flattens at the current level and all lower levels. RTL view: flattens to generic logic level Technology view: flattens to technology-cell level	Flattens only non-hidden transparent hierarchical instances; opaque and hidden hierarchical instances are not flattened.
RTL->Flattened View	Creates a new, unfiltered RTL schematic of the entire design, flattened to the level of generic logic cells.	
Technology->Flattened View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of technology cells.	

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

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

The most versatile commands, are Dissolve Instances and Flatten Current Schematic, which you can also use for selective flattening ([Selective Flattening, on page 125](#)).

See also:

- [Filtering Compared to Flattening, on page 126](#)
- [Selective Flattening, on page 125](#)

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 358](#) for descriptions):

- Use Hide Instances to hide instances that you do *not* want to flatten, then flatten the others (flattening operations do not recognize hidden instances). After flattening, you can Unhide Instances that are hidden.
- Flatten selected hierarchical instances using one of these commands:
 - If the current schematic is unfiltered, use Dissolve Instances.
 - If the schematic is filtered, use Dissolve Instances, followed by Flatten Current Schematic. In a filtered schematic, Dissolve Instances makes the selected instances transparent and Flatten Current Schematic flattens only transparent instances.

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

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

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

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

Although the result displayed is that of Step 2, you can view the intermediate result of Step 1 with View->Back. This is because the display history is erased before flattening (Step 1), and the result of Step 1 is added to the history as if you had viewed it.

Filtering Compared to Flattening

As a general rule, use filtering to examine your design, and flatten it only if you really need it. Here are some reasons to use filtering instead of flattening:

- Filtering before flattening is a more efficient use of computer time and memory. Creating a new view where everything is flattened can take considerable time and memory for a large design. You then filter anyway to remove the flattened logic you do not need.
- Filtering is selective. On the other hand, the default flattening operations are global: the entire design is flattened from the current level down.

Similarly, the inverse operation (UnFlatten Schematic) unflattens everything on the current schematic level.

- Flattening operations eliminate the *history* for the current view: You can not use View->Back after flattening. (You can, however, use UnFlatten Schematic to regenerate the unflattened schematic.).

See also:

- [RTL and Technology Views Popup Menus, on page 358](#)
- [Selective Flattening, on page 125](#)

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 128](#)
- [Critical Paths and the Slack Margin Parameter, on page 129](#)
- [Examining Critical Path Schematics, on page 130](#)

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

- [Watch Window, on page 44](#)
- [Log File, on page 254](#)
- [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 210](#)). 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 281](#)
- [Timing Reports, on page 259](#)
- [Log File, on page 254](#)

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 293](#)
- [HDL Analyst Command, on page 294](#)
- [Handling Negative Slack, on page 330](#) of the *User Guide*
- [Analyzing Timing in Schematic Views, on page 324](#) 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 123](#)
- [Filtering and Flattening Schematics, on page 122](#)

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 and physical constraints define regions and locations for placement-aware synthesis.

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

- [Constraint Types, on page 134](#)
- [Constraint Files, on page 135](#)
- [Timing Constraints, on page 137](#)
- [FDC Constraints, on page 140](#)
- [Methods for Creating Constraints, on page 141](#)
- [Constraint Translation, on page 143](#)
- [Constraint Checking, on page 146](#)
- [Database Object Search, on page 148](#)
- [Forward Annotation, on page 149](#)
- [Auto Constraints, on page 149](#)

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 (<code>create_clock</code>), clock groups (<code>set_clock_groups</code>), and timing exceptions like multicycle and false paths (<code>set_multicycle_path...</code>) See Timing Constraints, on page 137 for information on defining these constraints.
Design	Additional design goals that enhance or guide tool optimizations. Examples: Attributes and directives (<code>define_attribute</code> , <code>define_global_attribute</code>), I/O standards (<code>define_io_standard</code>), and compile points (<code>define_compile_point</code>).

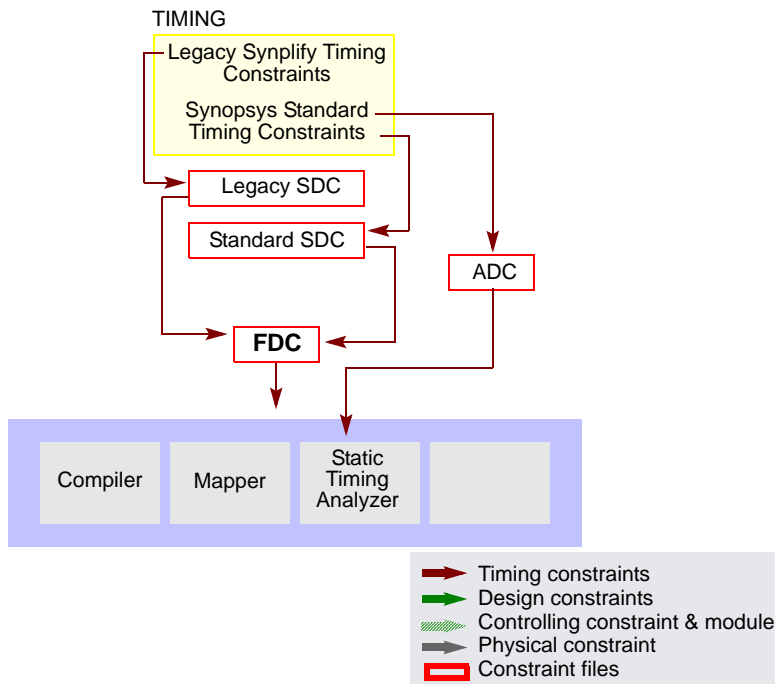
The easiest way to specify constraints is through the SCOPE interface. The tool saves timing and design constraints to an FDC file that you add to your project.

See Also

Constraint Files, on page 135	Overview of constraint files
Timing Constraints, on page 137	Overview of timing constraint definitions and FDC file generation.
SCOPE Constraints Editor, on page 151	Information about automatic generation of timing and design constraints.
Chapter 6, <i>Constraint Syntax</i>	Timing constraint syntax
Design Constraints, on page 237	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 137](#). 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	<code>create_clock,</code> <code>set_multicycle_delay ...</code>	Used for synthesis. Includes timing constraints that follow the Synopsys standard format as well as design constraints.
	Design constraints	<code>define_attribute,</code> <code>define_io_standard ...</code>	
ADC	Timing constraints for timing analysis	<code>create_clock,</code> <code>set_multicycle_delay ...</code>	Used with the stand-alone timing analyzer.
SDC (Synopsys Standard)	FPGA timing constraints	<code>create_clock,</code> <code>set_clock_latency,</code> <code>set_false_path ...</code>	Use <code>sd2f2dc</code> 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	<code>define_clock,</code> <code>define_false_path</code> <code>define_attribute,</code> <code>define_collection ...</code>	Use <code>sd2f2dc</code> 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 137](#)
- [FDC File Generation, on page 138](#)
- [Timing Constraint Precedence in Mixed Constraint Designs, on page 139](#)

Legacy SDC and Synopsys Standard SDC

Releases prior to G-2012.09M 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.

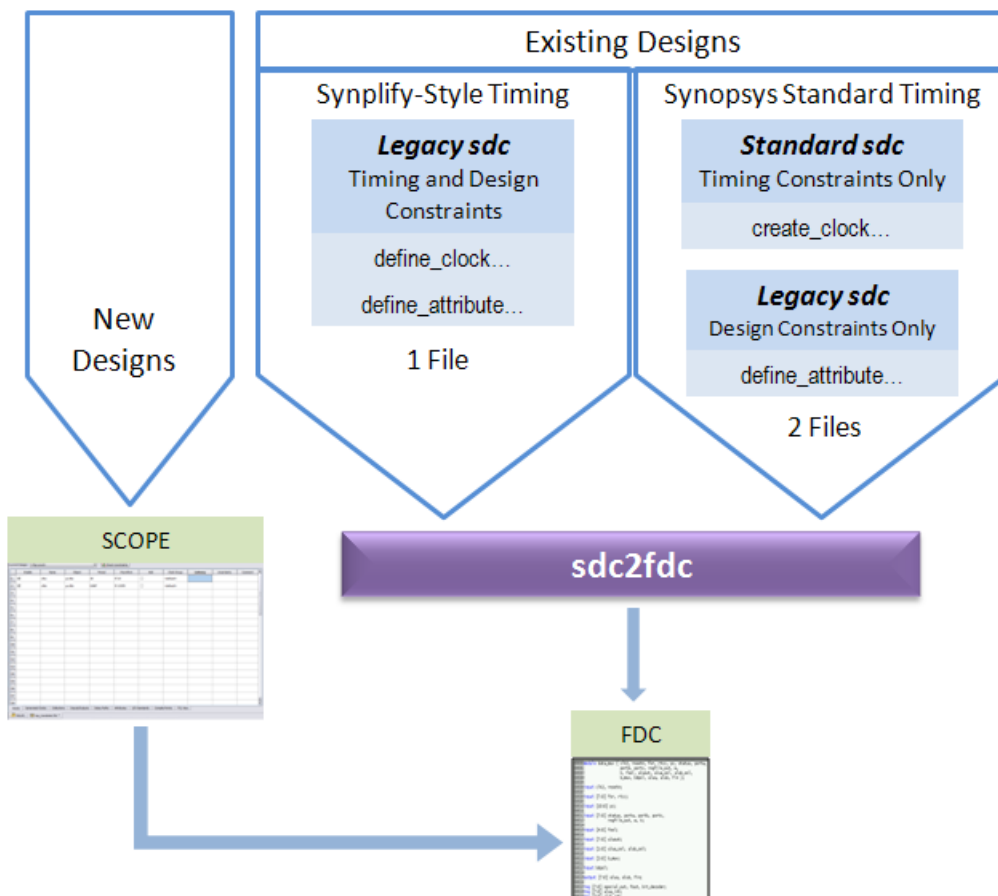
Starting with the G-2012.09M release, Synopsys standard timing constraint format has replaced the legacy-style constraint format, and a new FDC (FPGA design constraint) file consolidates both timing and design formats. As a result of these updates, there are some changes in the use model:

- Timing constraints in the legacy format are converted and included in an FDC file, which includes both timing and non-timing constraints. The file uses the Synopsys standard syntax for timing constraints (`create_clock`, `set_multicycle_path`...). The syntax for non-timing design constraints is unchanged (`define_attribute`, `define_io_standard`...).
- The SCOPE editor has been enhanced to support the timing constraint changes, so that new constraints can be entered correctly.
- For older designs, use the `sdcd2fdc` 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 [FPGA Timing Constraints, on page 200](#).

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 141](#) for details on generating constraint files.

Prior to release G-2012.09M, 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 158](#) for details on how to run `sdc2fdc`.

Methods for Creating Constraints

Constraints are passed to the synthesis environment in FDC files using Tcl command syntax.

New Designs

For new designs, you can specify constraints using any of the following methods:

Definition Method	Description
SCOPE Editor (fdc file)– Recommended	<p>Use this method to specify constraints wherever possible. The SCOPE editor automatically generates fdc constraints with the right syntax. You can use it for most constraints. See Chapter 5, SCOPE Constraints Editor, for information how to use SCOPE to automatically generate constraint syntax.</p> <p>Access: File->New->FPGA Design Constraints ...</p>
Manually-Entered Text Editor (fdc File, all other constraint files)	<p>You can manually enter constraints in a text file. Make sure to use the correct syntax for the timing and design commands.</p> <p>The SCOPE GUI includes a TCL View with an advanced text editor, where you can manually generate the constraint syntax. For a description of this view, see TCL View, on page 176.</p> <p>You can also open any constraint file in a text editor to modify it.</p>
Source Code Attributes/Directives (HDL files, cdc file)	<p>Directives must be entered in the source code because they affect the compiler. Do not include any other constraints in the source code, as this makes the source code less portable. In addition, you must recompile the design for the constraints to take effect.</p> <p>Attributes can be entered through the SCOPE interface, as they affect the mapper, not the compiler</p>
Automatic— First Pass	<p>Enable the Auto Constrain button in the Project view to have the tool automatically generate constraints based on inferred clocks. See Using Auto Constraints, on page 342 in the <i>User Guide</i> for details.</p> <p>Use this method as a quick first pass to get an idea of what constraints can be set.</p>

If there are multiple timing exception constraints on the same object, the software uses the guidelines described in [Conflict Resolution for Timing Exceptions, on page 194](#) to determine the constraint that takes precedence.

See Also

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

- [Chapter 6, Constraint Syntax](#)
- *Attribute Reference Manual*

Existing Designs

The SCOPE editor in this release does not save constraints to SDC files. For designs prior to G-2012.09M, 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 158l](#)) 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 198](#).

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.

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 143
- [sdc2fdc FPGA Design Constraint \(FDC\) File](#), on page 145
- [sdc2fdc](#), on page 54 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

Message Example	Underlying Problem
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.

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 268](#) 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
<code>_cck.rpt</code>	Lists the results of running the constraint checker (see Constraint Checking Report, on page 268).
<code>_cck_fdc_rpt</code>	Lists the wildcard expansion results of running the constraint checker for collections with the <code>get_*</code> and <code>all_*</code> object query commands using the <code>check_fdc_query</code> Tcl command. See check_fdc_query, on page 21 for more information.
<code>_scck.rpt</code>	Lists the results of running the constraint checker for collections with the <code>get_*</code> and <code>all_*</code> object query commands.
<code>.ta</code>	Reports timing analysis results (see Generating Custom Timing Reports with STA, on page 332).
<code>.srr</code> or <code>.htm</code>	Reports post-synthesis timing results as part of the text or HTML log file (see Timing Reports, on page 259 and Log File, on page 254).

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 148 and find, on page 92 .
Collections	define_collection, c_union...	Create, copy, evaluate, traverse, and filter collections. See Using Collections, on page 148 and Collection Commands, on page 116 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
Microsemi	_SDC.SDC

For information about how forward annotation is handled for Microsemi, see [Designing with Microsemi, on page 377](#).

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

CHAPTER 5

SCOPE Constraints Editor

The SCOPE (Synthesis Constraints OPTimization Environment[®]) editor automatically generates syntax for synthesis constraints. Enter information in the SCOPE tabs, panels, columns, and pull-downs to define constraints and parameter values. You can also drag and drop objects from the HDL Analyst UI to populate values in the constraint fields.

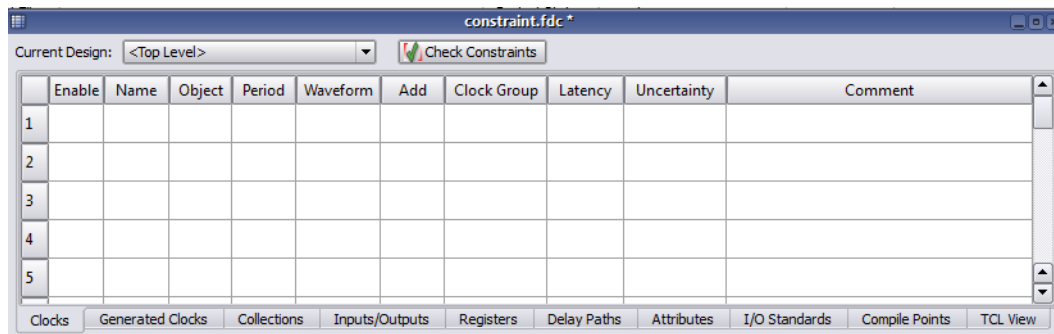
This interface creates Tcl-format *Synopsys Standard timing constraints* and *Synplify-style design constraints* and saves the syntax to an FPGA design constraints (FDC) file that can automatically be added to your synthesis project. See [Constraint Types, on page 134](#) for definitions of synthesis constraints.

Topics in this section include:

- [SCOPE User Interface, on page 152](#)
- [SCOPE Tabs, on page 153](#)
- [Industry I/O Standards, on page 178](#)
- [Delay Path Timing Exceptions, on page 182](#)
- [Specifying From, To, and Through Points, on page 187](#)
- [Conflict Resolution for Timing Exceptions, on page 194](#)

SCOPE User Interface

The SCOPE editor contains a number of panels for creating and managing timing constraints and design attributes. This GUI offers the easiest way to create constraint files for your project. The syntax is saved to a file using an FDC extension and can be included in your design project.



From this editor, you specify timing constraints for clocks, ports, and nets as well as design constraints such as attributes, collections, and compile points. However, you cannot set black-box constraints from the SCOPE window.

To bring up the editor, use one of the following methods from the Project view:

- For a new file (the project file is open and the design is compiled):
 - Choose File->New-> FPGA Design Constraints; select FPGA Constraint File (SCOPE).
 - Click the SCOPE icon in the toolbar; select FPGA Constraint File (SCOPE).
- You can also open the editor using an existing constraint file. Double-click on the constraint file (FDC), or use File->Open, specifying the file type as FPGA Design Constraints File (*.fdc).

See [Using the SCOPE Editor, on page 114](#) in the *User Guide*.

SCOPE Tabs

Here is a summary of the constraints created through the SCOPE editor:

SCOPE Panel	See ...
Clocks	Clocks, on page 153
Generated Clocks	Generated Clocks, on page 159
Collections	Collections, on page 161
Inputs/Outputs	Inputs/Outputs, on page 163
Registers	Registers, on page 167
Delay Paths	Delay Paths, on page 168
Attributes	Attributes, on page 170
I/O Standards	I/O Standards, on page 171
Compile Points	Compile Points, on page 173
TCL View	TCL View, on page 176

If you choose an object from a SCOPE pull-down menu, it has the appropriate prefix appended automatically. If you drag and drop an object from an RTL view, for example, make sure to add the prefix appropriate to the language used for the module. See [Naming Rule Syntax Commands, on page 234](#) for details.

Clocks

Use the Clocks panel of the SCOPE spreadsheet to define a signal as a clock.

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	Uncertainty	Comment
1										
2										
3										
4										
Clocks										

The Clocks panel includes the following options:

Field	Description
Name	<p>Specifies the clock object name.</p> <p>Clocks can be defined on the following objects:</p> <ul style="list-style-type: none">• Pins• Ports• Nets <p>For virtual clocks, the field must contain a unique name not associated with any port, pin, or net in the design.</p>
Period	<p>Specifies the clock period in nanoseconds. This is the minimum time over which the clock waveform repeats. The period must be greater than zero.</p>
Waveform	<p>Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time in the list is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise and then fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of period/2.</p>
Add Delay	<p>Specifies whether to add this delay to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you use this option, you must also specify the clock, and clocks with the same source must have different names.</p>

Field	Description
Clock Group	Assigns clocks to asynchronous clock groups. The clock grouping is inclusionary (for example, clk2 and clk3 can each be related to clk1 without being related to each other). For details, see Clock Groups, on page 155 .
Latency	Specifies the clock latency applied to clock ports and clock aliases. Applying the latency constraint on a port can be used to model the off-chip clock delays in a multichip environment. Clock latency can only: <ul style="list-style-type: none"> • Apply to clocks defined on input ports. • Be used for source latency. • Apply to port clock objects.
Uncertainty	Specifies the clock uncertainty (skew characteristics) of the specified clock networks. You can only apply latency to clock objects.

Clock Groups

Clock grouping is associative; two clocks can be asynchronous to each other but both can be synchronous with a third clock.

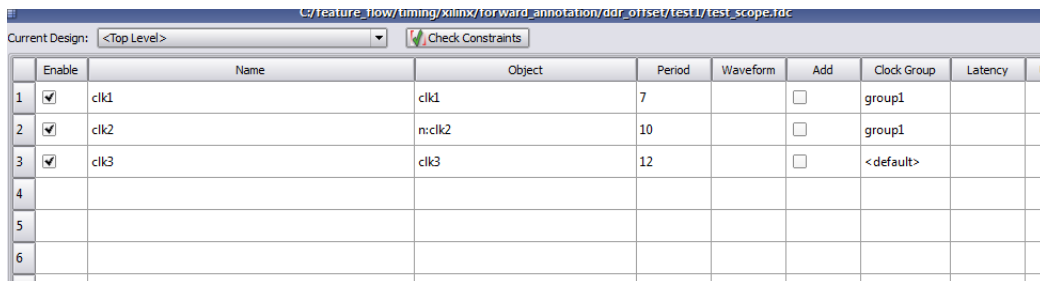
The SCOPE GUI prompts you for a clock group for each clock that you define. By default, the tool assigns all clocks to the default clock group. When you add a name that differs from the default clock group name, the clock is assigned its own clock group and is asynchronous to the default clock group as well as all other named clock groups.

This section presents scenarios for defining clocks and includes the following examples:

- [Example 1 – SCOPE Definition](#)
- [Example 2 – Equivalent Tcl Syntax](#)
- [Example 3 – Establish Clock Relationships](#)
- [Example 4 – Using a Single Group Option](#)
- [Example 5 – Legacy Clock Grouping](#)

Example 1 – SCOPE Definition

A design has three clocks, clk1, clk2, clk3. You want clk1 and clk2 to be in the same clock group—synchronous to each other but asynchronous to clk3. You can apply this clock definition by adding a name in the Clock Group column, as shown below:



The screenshot shows the SCOPE Constraints Editor window. The title bar indicates the file path: C:/feature_flow/timing/xilinx/forward_annotation/dor_onset/test1/test_scope.tdc. The 'Current Design' dropdown is set to '<Top Level>'. A 'Check Constraints' button is visible. Below is a table with columns: Enable, Name, Object, Period, Waveform, Add, Clock Group, Latency, and U. The table contains three rows of clock definitions.

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	U
1	<input checked="" type="checkbox"/>	clk1	clk1	7		<input type="checkbox"/>	group1		
2	<input checked="" type="checkbox"/>	clk2	n:clk2	10		<input type="checkbox"/>	group1		
3	<input checked="" type="checkbox"/>	clk3	clk3	12		<input type="checkbox"/>	<default>		
4									
5									
6									

This definition assigns clk1 and clk2 to clock group group1, synchronous to each other and asynchronous to clk3. The equivalent Tcl command for this appears in the text editor window as follows:

```
set_clock_groups -derive -asynchronous -name {group1}
                  -group {{c:clk1} {c:clk2}}
```

Example 2 – Equivalent Tcl Syntax

A design has three clocks: clk1, clk2, clk3. Use the following commands to set clk2 synchronous to clk3, but asynchronous to clk1:

```
set_clock_groups -asynchronous -group [get_clocks {clk3 clk2}]
set_clock_groups -asynchronous -group [get_clocks {clk1}]
```

Example 3 – Establish Clock Relationships

A design has the following clocks defined:

```
create_clock -name {clk_a} {p:clk_a} -period 10 -waveform {0 5.0}
create_clock -name {clk_b} {p:clk_b} -period 20 -waveform {0 10.0}
create_clock -name {my_sys} {p:sys_clk} -period 200 -waveform {0
100.0}
```

You want to define `clk_a` and `clk_b` as asynchronous to each other and `clk_a` and `clk_b` as synchronous to `my_sys`.

For the tool to establish these relationships, multiple `-group` options are needed in a single `set_clock_groups` command. Clocks defined by the first `-group` option are asynchronous to clocks in the subsequent `-group` option. Therefore, you can use the following syntax to establish the relationships described above:

```
set_clock_groups -asynchronous -group [get_clocks {clk_a}]
                  -group [get_clocks {clk_b}]
```

Example 4 – Using a Single Group Option

`set_clock_groups` has a unique behavior when a single `-group` option is specified in the command. For this example, the following constraint specifications are applied:

```
set_clock_groups -asynchronous -name {default_clkgroup_0} -group
[get_clocks {clk_a my_sys}]

set_clock_groups -asynchronous -name {default_clkgroup_1} -group
[get_clocks {clk_b my_sys}]
```

The first statement assigns `clk_a` AND `my_sys` as asynchronous to `clk_b`, and the second statement assigns `clk_b` AND `my_sys` as asynchronous to `clk_a`. Therefore, with this specification, all three clocks are established as asynchronous to each other.

Example 5 – Legacy Clock Grouping

This section shows how the legacy clock group definitions (Synplify-style timing constraints) are converted to the Synopsys standard timing syntax (FDC). Legacy clock grouping can be represented through Synopsys standard constraints, but the multi-grouping in the Synopsys standard constraints cannot be represented in legacy constraints.

For example, the following table shows legacy clock definitions and their translated FDC equivalents:

Legacy Definition	<pre>define_clock -name {clka} {p:clka} -period 10 -clockgroup default_clkgroup_0 define_clock -name {clkb} {p:clkb} -freq 150 -clockgroup default_clkgroup_1 define_clock -name {clkc} {p:clkc} -freq 200 -clockgroup default_clkgroup_1</pre>
FDC Definition	<pre>##### BEGIN Clocks - (Populated from SCOPE tab, do not edit) create_clock -name {clka} {p:clka} -period 10 -waveform {0 5.0} create_clock -name {clkb} {p:clkb} -period 6.667 -waveform {0 3.3335} create_clock -name {clkc} {p:clkc} -period 5.0 -waveform {0 2.5} set_clock_groups -derive -name default_clkgroup_0 -asynchronous -group {c:clka} set_clock_groups -derive -name default_clkgroup_1 -asynchronous -group {c:clkb c:clkc} ##### END Clocks</pre>

The `create_generated_clock` constraints used in legacy SDC are preserved in FDC. The `-derive` option directs the `create_generated_clock` command to inherit the `-source` clock group. This behavior is unique to FDC and is an extension of the Synopsys SDC standard functionality.

See Also

For equivalent Tcl syntax, see the following sections:

- [create_clock, on page 202](#)
- [set_clock_groups, on page 209](#)
- [set_clock_latency, on page 213](#)
- [set_clock_uncertainty, on page 216](#)

For information about other SCOPE panels, see [SCOPE Tabs, on page 153](#).

Generated Clocks

Use the Generated Clocks panel of the SCOPE spreadsheet to define a signal as a generated clock. The equivalent Tcl constraint is `create_generated_clock`; its syntax is described in [create_generated_clock](#), on page 204.

	Enable	Name	Source	Object	Master Clock	Generate Type	Generate Parameters	Generate Modifier	Modifier Parameters	Invert	Add	Comment
1	<input type="checkbox"/>											
2	<input type="checkbox"/>											
3	<input type="checkbox"/>											
4	<input type="checkbox"/>											

Generated Clocks

The Generated Clocks panel includes the following options:

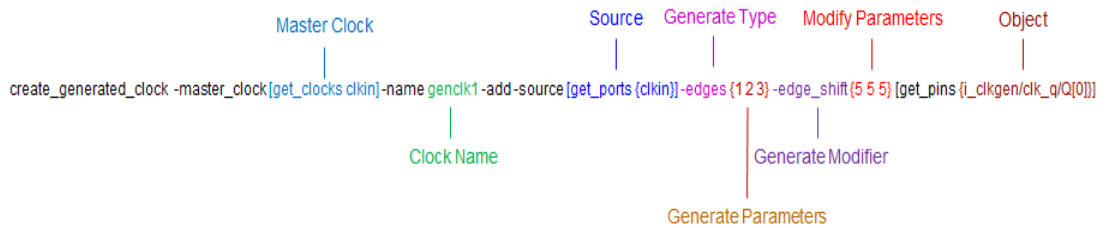
Field	Description
Name	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the source.
Source	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.
Object	Generated clocks can be defined on the following objects: <ul style="list-style-type: none"> • Pins • Ports • Nets • Instances—Where instances have only one output
Master Clock	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.

Field	Description
Generate Type	<p>Specifies any of the following:</p> <p>edges – Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.</p> <p>divide_by – Specifies the frequency division factor. If the divide factor value is 2, the generated clock period is twice as long as the master clock period.</p> <p>multiply_by – Specifies the frequency multiplication factor. If the multiply factor value is 3, the generated clock period is one-third as long as the master clock period.</p>
Generate Parameters	Specifies integers that define the type of generated clock.
Generate Modifier	Defines the secondary characteristics of the generated clock.
Modify Parameters	Defines modifier values of the generated clock.
Invert	Specifies whether to use invert – Inverts the generated clock signal (in the case of frequency multiplication and division).
Add	Either add this clock to the existing clock or overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also specify the clock and master clock. The clocks with the same source must have different names.

Examples

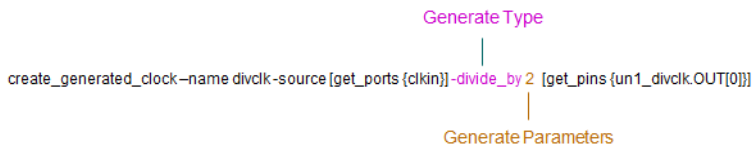
In the following example, the generated clock `genclk1` is created with the same frequency as the source clock `clkin`, but its phase is shifted by 180. Each of the edges of the generated clock shifts by 5 ns, which is specified by the `-edges` and `-edge_shift` options.

Example 1



For this example, a generated clock is created with half the frequency of the source clock.

Example 2



For more information about other SCOPE options, see [SCOPE Tabs](#), on [page 153](#).

Collections

The Collections tab allows you to set constraints for a group of objects you have defined as a collection with the Tcl command. For details, see [Creating and Using SCOPE Collections](#), on [page 149](#) of the *User Guide*.

	Enabled	Collection Name	Command	Command Arguments	Comment
1					
2					
3					
4					
...					

Collections

Field	Description
Enable	Enables the row.
Name	Enter the collection name.
Command	Select a collection creation command from the drop-down menu. See Collection Commands, on page 162 for descriptions of the commands.
Comment	Enter comments that are included in the constraints file.

You can crossprobe the collection results to an HDL Analyst view. To do this, right-click in the SCOPE cell and select the option **Select in Analyst**.

Collection Commands

You can use the collection commands on collections or Tcl lists. Tcl lists can be just a single element long.

To ...	Use this command ...
Create a collection	<p><code>set modules</code> To create and save a collection, assign it to a variable. You can also use this command to create a collection from any combination of single elements, TCL lists and collections:</p> <pre>set modules [define_collection {v:top} {v:cpu} \$mycoll \$mylist]</pre> <p>Once you have created a collection, you can assign constraints to it in the SCOPE interface.</p>
Copy a collection	<p><code>set modules_copy \$modules</code> This copies the collection, so that any change to <code>\$modules</code> does not affect <code>\$modules_copy</code>.</p>
Evaluate a collection	<p><code>c_print</code> This command returns all objects in a column format. Use this for visual inspection.</p> <p><code>c_list</code> This command returns a Tcl list of objects. Use this to convert a collection to a list. You can manipulate a Tcl list with standard Tcl list commands.</p>
Concatenate a list to a collection	<p><code>c_union</code></p>

To ...	Use this command ...
Identify differences between lists or collections	<code>c_diff</code> Identifies differences between a list and a collection or between two or more collections. Use the <code>-print</code> option to display the results.
Identify objects common to a list and a collection	<code>c_intersect</code> Use the <code>-print</code> option to display the results.
Identify objects common to two or more collections	<code>c_sub</code> Use the <code>-print</code> option to display the results.
Identify objects that belong exclusively to only one list or collection	<code>c_symdiff</code> Use this to identify unique objects in a list and a collection, or two or more collections. Use the <code>-print</code> option to display the results.

For information about all SCOPE panels, see [SCOPE Tabs, on page 153](#).

Inputs/Outputs

The Inputs/Outputs panel models the interface of the FPGA with the outside environment. You use it to specify delays outside the device.

	Enable	Delay Type	Port	Rise	Fall	Max	Min	Clock	Clock Fall	Add Delay	Value	Comment
1												
2												
3												
4												

Inputs/Outputs

The Inputs/Outputs panel includes the following options:

Field	Description
Delay Type	Specifies whether the delay is an input or output delay.
Port	Specifies the name of the port.
Rise	Specifies that the delay is relative to the rising transition on specified port. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.
Fall	Specifies that the delay is relative to the falling transition on specified port Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
Max	Specifies that the delay value is relative to the longest path. Note: The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
Min	Specifies that the delay value is relative to the shortest path. Note: The synthesis tool does not optimize for hold time violations and only reports -min delay values in the <code>synlog/topLevel_fpga_mapper.srr_Min</code> timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.
Clock	Specifies the name of a clock for which the specified delay is applied. If you specify the clock fall, you must also specify the name of the clock.
Clock Fall	Specifies that the delay relative to the falling edge of the clock. For examples, see Input Delays, on page 165 and Output Delays, on page 165 .
Add Delay	Specifies whether to add delay information to the existing input delay or overwrite the input delay. For examples, see Input Delays, on page 165 and Output Delays, on page 165 .
Value	Specifies the delay path value.

Input Delays

Here is how this constraint applies for input delays:

- **Clock Fall** – The default is the rising edge or rising transition of a reference pin. If you specify clock fall, you must also specify the name of the clock.
- **Add Delay** – Use this option to capture information about multiple paths leading to an input port relative to different clocks or clock edges.

For example, `set_input_delay 5.0 -max -rise -clock phi1 {A}` removes all maximum rise input delay from A, because the `-add_delay` option is not specified. Other input delays with different clocks or with `-clock_fall` are removed.

In this example, the `-add_delay` option is specified as `set_input_delay 5.0 -max -rise -clock phi1 -add_delay {A}`. If there is an input maximum rise delay for A relative to clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is maximum rise input delay relative to a different clock or different edge of the same clock, it remains with the new delay.

Output Delays

Here is how this constraint applies for output delays:

- **Clock Fall** – If you specify clock fall, you must also specify the name of the clock.
- **Add Delay** – By using this option, you can capture information about multiple paths leading from an output port relative to different clocks or clock edges.

For example, the `set_output_delay 5.0 -max -rise -clock phi1 {OUT1}` command removes all maximum rise output delays from OUT1, because the `-add_delay` option is not specified. Other output delays with a different clock or with the `-clock_fall` option are removed.

In this example, the `-add_delay` option is specified: `set_output_delay 5.0 -max -rise -clock phi1 -add_delay {Z}`. If there is an output maximum rise delay for Z relative to the clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is a maximum rise

output delay relative to a different clock or different edge of the same clock, it remains with the new delay.

Priority of Multiple I/O Constraints

You can specify multiple input and output delays constraints for the same I/O port. This is useful for cases where a port is driven by or feeds multiple clocks. The priority of a constraint and its use in your design is determined by a few factors:

- The software applies the tightest constraint for a given clock edge, and ignores all others. All applicable constraints are reported in the timing report.
- You can apply I/O constraints on three levels, with the most specific overriding the more global:
 - Global (top-level netlist), for all inputs and outputs
 - Port-level, for the whole bus
 - Bit-level, for single bits

If there are two bit constraints and two port constraints, the two bit constraints override the two port constraints for that bit. The other bits get the two port constraints. For example, take the following constraints:

```
a[3:0] 3 clk1:r
a[3:0] 3 clk2:r
a[0] 2 clk1:r
```

In this case, port a[0] only gets one constraint of 2 ns. Ports a[1], a[2], and a[3] get two constraints of 3 ns each.

- If at any given level (bit, port, global) there is a constraint with a reference clock specified, then any constraint without a reference clock is ignored. In this example, the 1 ns constraint on port a[0] is ignored.

```
a[0] 2 clk1:r
a[0] 1
```

See Also

For equivalent Tcl syntax, see:

- [set_input_delay, on page 221](#)
- [set_output_delay, on page 230](#)

For information about all SCOPE panels, see [SCOPE Tabs, on page 153](#).

Registers

This panel lets the advanced user add delays to paths feeding into/out of registers, in order to further constrain critical paths. Use this constraint to speed up the paths feeding a register. See [set_reg_input_delay, on page 233](#), and [set_reg_output_delay, on page 234](#) for the equivalent Tcl commands.

The Registers SCOPE panel includes the following fields:

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Delay Type	(Required) Specifies whether the delay is an input or output delay.
Register	(Required) Specifies the name of the register. If you have initialized a compiled design, you can choose from the pull-down list.
Route	(Required) Improves the speed of the paths to or from the register by the given number of nanoseconds. The value shrinks the effective period for the constrained registers without affecting the clock period that is forward-annotated to the place-and-route tool.
Comment	Lets you enter comments that are included in the constraints file.

Delay Paths

Use the Delay Paths panel to define the timing exceptions.

	Enable	Delay Type	From	Through	To	Max Delay	Setup	Start/End	Cycles	Comment
1	<input type="checkbox"/>	<div>▼ Multicycle False Max Delay Reset Path Datapath Only</div>					<input type="checkbox"/>			
2										
3										
4										

Delay Paths

The Path Delay panel includes the following options:

Field	Description
Delay Type	<p>Specifies the type of delay path you want the synthesis tool to analyze. Choose one of the following types:</p> <ul style="list-style-type: none"> • Multicycle • False • Max Delay • Reset Path • Datapath Only
From	<p>Starting point for the path. From points define timing start points and can be defined for clocks (c:), registers (i:), top-level input or bi-directional ports (p:), black box output pins (i:) or sequential cell clock pins. For details, see the following:</p> <ul style="list-style-type: none"> • Defining From/To/Through Points for Timing Exceptions • Naming Rule Syntax Commands, on page 234
Through	<p>Specifies the intermediate points for the timing exception. Intermediate points can be combinational nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). If you click the arrow in a column cell, you open the Product of Sums (POS) interface where you can set through constraints. For details, see the following:</p> <ul style="list-style-type: none"> • Product of Sums Interface • Defining From/To/Through Points for Timing Exceptions • Naming Rule Syntax Commands, on page 234

Field	Description
To	Ending point of the path. To points must be timing end points and can be defined for clocks (c:), registers (i:), top-level output or bi-directional ports (p:), or black box input pins (i:). For details, see the following: <ul style="list-style-type: none">• Defining From/To/Through Points for Timing Exceptions• Naming Rule Syntax Commands, on page 234
Max Delay	Specifies the maximum delay value for the specified path in nanoseconds.
Setup	Specifies the setup (maximum delay) calculations used for specified path.
Start/End	Used for multicycle paths with different start and end clocks. This option determines the clock period to use for the multiplicand in the calculation for clock distance. If you do not specify a start or end clock, the end clock is the default.
Cycles	Specifies the number of cycles required for the multicycle path.

See Also

- For equivalent Tcl syntax, see:
 - [set_multicycle_path, on page 226](#)
 - [set_false_path, on page 218](#)
 - [set_max_delay, on page 223](#)
 - [reset_path, on page 207](#)
- For more information on timing exception constraints and how the tool resolves conflicts, see:
 - [Delay Path Timing Exceptions, on page 182](#)
 - [Conflict Resolution for Timing Exceptions, on page 194](#)
- For information about all SCOPE panels, see [SCOPE Tabs, on page 153](#).

Attributes

You can assign attributes directly in the editor.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	mme
1	<input checked="" type="checkbox"/>	output_port	<global>	syn_noclockbuf				
2	<input checked="" type="checkbox"/>			syn_clean_reset				
3	<input checked="" type="checkbox"/>			syn_dspstyle				
4	<input checked="" type="checkbox"/>			syn_edif_bit_format				
5	<input checked="" type="checkbox"/>			syn_edif_scalar_format				
				syn_forwar...onstraints				
				syn_multstyle				
				syn_netlist_hierarchy				
				syn_noarrayports				
				syn_noclockbuf				
				syn_ramstyle				

Here are descriptions for the Attributes columns:

Column	Description
Enabled	(Required) Turn this on to enable the constraint.
Object Type	Specifies the type of object to which the attribute is assigned. Choose from the pull-down list, to filter the available choices in the Object field.
Object	(Required) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field.
Attribute	<p>(Required) Specifies the attribute name. You can choose from a pull-down list that includes all available attributes for the specified technology. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the Synopsys FPGA synthesis tool filters the available choices in the Object field. You must select an attribute before entering a value.</p> <p>If a valid attribute does not appear in the pull-down list, simply type it in this field and then apply appropriate values.</p>
Value	(Required) Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate.

Val Type	Specifies the kind of value for the attribute. For example, string or boolean.
Description	Contains a one-line description of the attribute.
Comment	Lets you enter comments about the attributes.

Enter the appropriate attributes and their values, by clicking in a cell and choosing from the pull-down menu.

To specify an object to which you want to assign an attribute, you may also drag-and-drop it from the RTL or Technology view into a cell in the Object column. After you have entered the attributes, save the constraint file and add it to your project.

See Also

- For more information on specifying attributes, see [How Attributes and Directives are Specified, on page 8](#).
- For information about all SCOPE panels, see [SCOPE Tabs, on page 153](#).

I/O Standards

You can specify a standard I/O pad type to use in the design. Define an I/O standard for any port appearing in the I/O Standards panel.

	Enabled	Port	Type	I/O Standard	DCI	DV2	Slew Rate	Drive Strength	Termination	Description
1	<input checked="" type="checkbox"/>	<input default>	input	LVCMOS_15			fast	8	pullup	1.5 volt - C...
2	<input checked="" type="checkbox"/>	<output default>	output							
3	<input type="checkbox"/>	<bidir default>	bidir							
4	<input type="checkbox"/>	resetn	input							

Field	Description
Enabled	(Required) Turn this on to enable the constraint, or off to disable a previous constraint.
Port	(Required) Specifies the name of the port. If you have initialized a compiled design, you can select a port name from the pull-down list. The first two entries let you specify global input and output delays, which you can then override with additional constraints on individual ports.
Type	(Required) Specifies whether the delay is an input or output delay.
I/O Standard	Supported I/O standards by Synopsys FPGA products. See Industry I/O Standards, on page 178 for a description of the standards.
Slew Rate Drive Strength Termination Power Schmitt	The values for these parameters are based on the selected I/O standard.
Description	Describes the selected I/O Standard.
Comment	Enter comments about an I/O standard.

See Also

- The Tcl equivalent of this constraint is [define_io_standard](#).
- For information about all SCOPE panels, see [SCOPE Tabs, on page 153](#).

Compile Points

Use the Compile Points panel to specify compile points in your design, and to enable/disable them. This panel, available only if the device technology supports compile points, is used to define a top-level constraint file.

	Enabled	Module	Type	Comment
1	<input checked="" type="checkbox"/>		<div><div></div><div>locked</div><div>locked,partition</div><div>soft</div><div>hard</div><div>black_box</div></div>	
2	<input type="checkbox"/>			
3	<input type="checkbox"/>			
4	<input type="checkbox"/>			

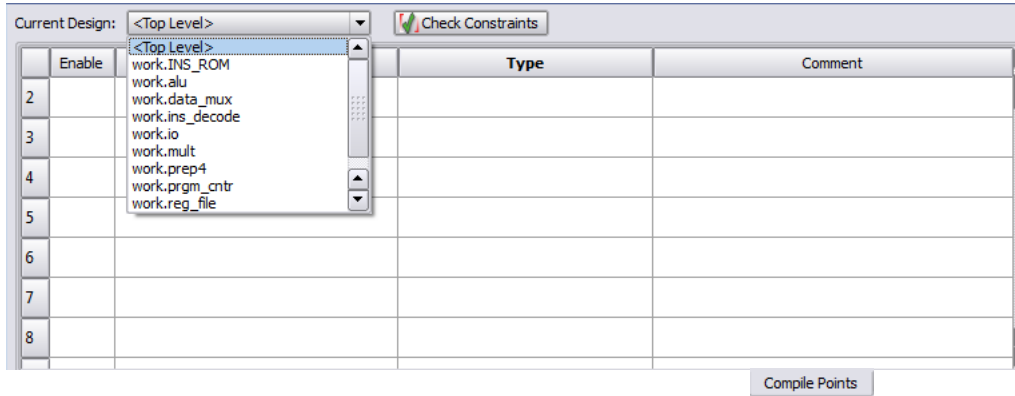
Compile Points

Here are the descriptions of the fields in the Compile Points panel.

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Module	(Required) Specifies the name of the compile-point module. You must specify a view module, with a v: prefix to identify the module as a view. For example: v:alu.
Type	<p>(Required) Specifies the type of compile point:</p> <ul style="list-style-type: none"> locked (default) – no timing reoptimization is done on the compile point. The hierarchical interface is unchanged and an interface logic model is constructed for the compile point. soft – compile point is included in the top-level synthesis, boundary optimizations can occur. hard – compile point is included in the top-level synthesis, boundary optimizations can occur, however, the boundary remains unchanged. Although, the boundary is not modified, instances on both sides of the boundary can be modified using top-level constraints. <p>For details, see Compile Point Types, on page 419 in the <i>User Guide</i>.</p>
Comment	Lets you enter a comment about the compile point.

Constraints for Compile Points

You can set constraints at the top-level or for modules to be used as the compile points from the Current Design pull-down menu shown below. Use the Compile Points tab to select compile points and specify their types.

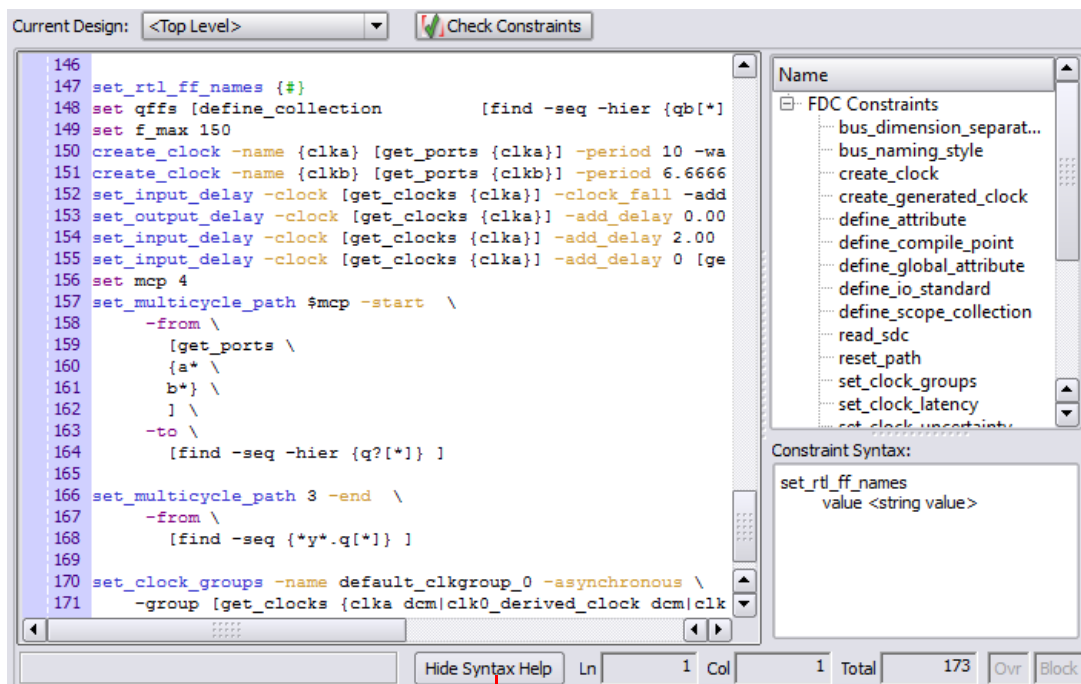


See Also

- The Tcl equivalent is [define_compile_point](#).
- For more information on compile points and using the Compile Points panel, see [Synthesizing Compile Points, on page 433](#) in the *User Guide*.
- For information about all SCOPE panels, see [SCOPE Tabs, on page 153](#).

TCL View

The TCL View is an advanced text file editor for defining FPGA timing and design constraints.



Click on **Hide Syntax Help**
to close this browser

This text editor provides the following capabilities:

- Uses dynamic keyword expansion and tool tips for commands that
 - Automatically completes the command from a popup list
 - Displays complete command syntax as a tool tip
 - Displays parameter options for the command from a popup list
 - Includes a keyword command syntax help

- Checks command syntax and uses color indicators that
 - Validate commands and command syntax
 - Identifies FPGA design constraints and SCOPE legacy constraints
- Allows for standard editor commands, such as copy, paste, comment/un-comment a group of lines, and highlighting of keywords

For information on how to use this Tcl text editor, see [Using the TCL View of SCOPE GUI, on page 127](#).

See Also

- For Tcl timing constraint syntax, see [FPGA Timing Constraints, on page 200](#).
- For Tcl design constraint syntax, see [Design Constraints, on page 237](#).
- You can also use the SCOPE editor to set attributes. See [How Attributes and Directives are Specified, on page 8](#) for details.

Industry I/O Standards

The synthesis tool lets you specify a standard I/O pad type to use in your design. You can define an I/O standard for any port supported from the industry standard and proprietary I/O standards.

For industry I/O standards, see [Industry I/O Standards, on page 179](#).

For vendor-specific I/O standards, see [Microsemi I/O Standards, on page 379](#).

Industry I/O Standards

The following table lists industry I/O standards.

I/O Standard	Description
AGP1X	Intel Corporation Accelerated Graphics Port
AGP2X	Intel Corporation Accelerated Graphics Port
BLVDS_25	Bus Differential Transceiver
CTT	Center Tap Terminated - EIA/JEDEC Standard JESD8-4
DIFF_HSTL_15_Class_I	1.5 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
DIFF_HSTL_15_Class_II	1.5 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
DIFF_HSTL_18_Class_I	1.8 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-9A
DIFF_HSTL_18_Class_II	1.8 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-9A
DIFF_SSTL_18_Class_II	1.8 volt - Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-6
DIFF_SSTL_2_Class_I	2.5 volt - Pseudo Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9A
DIFF_SSTL_2_Class_II	2.5 volt - Pseudo Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9A
GTL	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+	Gunning Transceiver Logic Plus
GTL25	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+25	Gunning Transceiver Logic Plus
GTL33	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+33	Gunning Transceiver Logic Plus

I/O Standard	Description
HSTL_12	1.2 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_15_Class_II	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_I	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_II	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_III	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_IV	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_I	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_II	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_III	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_IV	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HyperTransport	2.5 volt - Hypertransport - HyperTransport Consortium

I/O Standard	Description
LVC MOS_12	1.2 volt - EIA/JEDEC Standard JESD8-16
LVC MOS_15	1.5 volt - EIA/JEDEC Standard JESD8-7
LVC MOS_18	1.8 volt - EIA/JEDEC Standard JESD8-7
LVC MOS_25	2.5 volt - EIA/JEDEC Standard JESD8-5
LVC MOS_33	3.3 volt CMOS - EIA/JEDEC Standard JESD8-B
LVC MOS_5	5.0 volt CMOS
LVDS	Differential Transceiver - ANSI/TIA/EIA-644-95
LVDSEXT_25	Differential Transceiver
LVPECL	Differential Transceiver - EIA/JEDEC Standard JESD8-2
LVTTL	3.3 volt TTL - EIA/JEDEC Standard JESD8-B
MINI_LVDS	Mini Differential Transceiver
PCI33	3.3 volt PCI 33MHz - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCI66	3.3 volt PCI 66MHz - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCI-X_133	3.3 volt PCI-X - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCML	3.3 volt - PCML
PCML_12	1.2 volt - PCML
PCML_14	1.4 volt - PCML
PCML_15	1.5 volt - PCML
PCML_25	2.5 volt - PCML
RSDS	Reduced Swing Differential Signalling
SSTL_18_Class_I	1.8 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-15
SSTL_18_Class_II	1.8 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-15
SSTL_2_Class_I	2.5 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9B
SSTL_2_Class_II	2.5 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9B
SSTL_3_Class_I	3.3 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-8
SSTL_3_Class_II	3.3 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-8
ULVDS_25	Differential Transceiver

Delay Path Timing Exceptions

For details about the following path types, see:

- [Multicycle Paths, on page 182](#)
- [False Paths, on page 185](#)

Multicycle Paths

Multicycle paths lets you specify paths with multiple clock cycles. The following table defines the parameters for this constraint. For the equivalent Tcl constraints, see [set_multicycle_path, on page 226](#). This section describes the following:

- [Multi-cycle Path with Different Start and End Clocks, on page 182](#)
- [Multicycle Path Examples, on page 183](#)

Multi-cycle Path with Different Start and End Clocks

The `start/end` option determines the clock period to use for the multiplicand in the calculation for required time. The following table describes the behavior of the multi-cycle path constraint using different start and end clocks. In all equations, n is number of clock cycles, and *clock_distance* is the default, single-cycle relationship between clocks that is calculated by the tool.

Basic required time for a multi-cycle path	$\text{clock_distance} + [(n-1) * \text{end_clock_period}]$
Required time with no end clock defined	$\text{clock_distance} + [(n-1) * \text{global_period}]$
Required time with <code>-start</code> option defined	$\text{clock_distance} + [(n-1) * \text{start_clock_period}]$
Required time with no start clock defined	$\text{clock_distance} + [(n-1) * \text{global_period}]$

If you do not specify a start or end option, by default the end clock is used for the constraint. Here is an example:

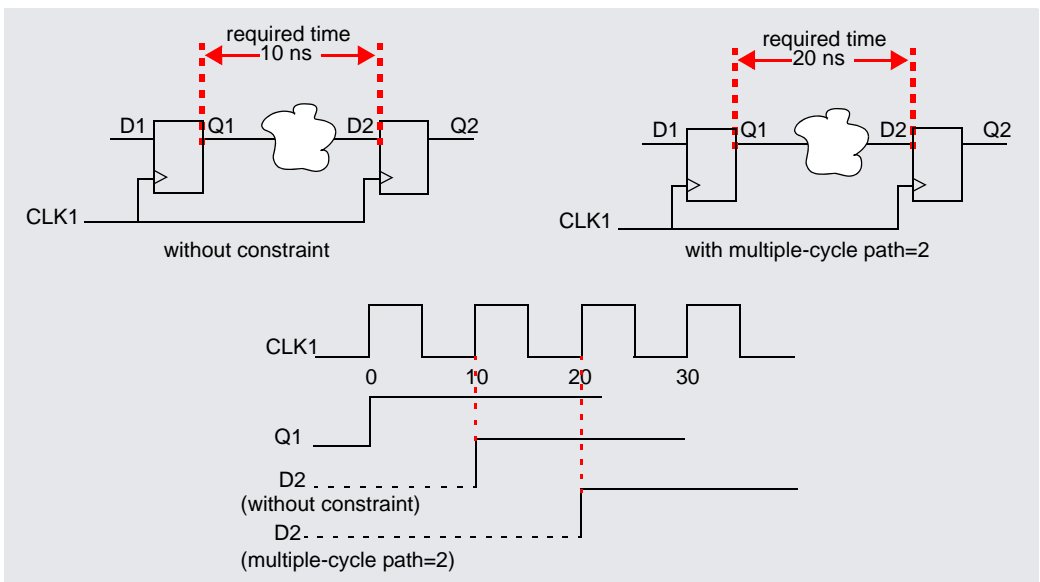
	Enabled	Delay Type	From	To	Through	Start/End	Cycles	Max Delay(ns)	Comment
1	<input checked="" type="checkbox"/>	Multicycle				End			
2	<input type="checkbox"/>					Start			
3	<input type="checkbox"/>					End			
4	<input type="checkbox"/>								

Delay Paths

Multicycle Path Examples

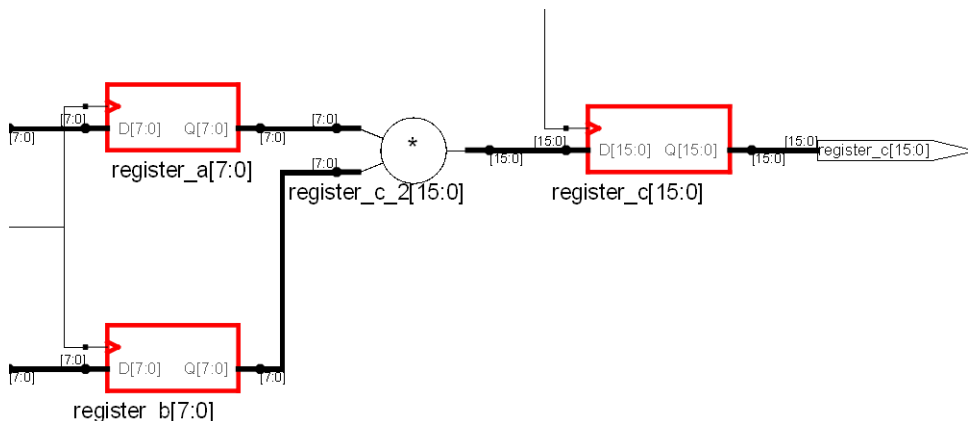
Multicycle Path Example 1

If you apply a multicycle path constraint from D1 to D2, the allowed time is $\#cycles \times \text{normal time between D1 and D2}$. In the following figure, CLK1 has a period of 10 ns. The data in this path has only one clock cycle before it must reach D2. To allow more time for the signal to complete this path, add a multiple-cycle constraint that specifies two clock cycles (10×2 or 20 ns) for the data to reach D2.



Multicycle Path Example 2

The design has a multiplier that multiplies signal_a with signal_b and puts the result into signal_c. Assume that signal_a and signal_b are outputs of registers register_a and register_b, respectively. The RTL view for this example is shown below. On clock cycle 1, a state machine enables an input enable signal to load signal_a into register_a and signal_b into register_b. At the beginning of clock cycle 2, the multiply begins. After two clock cycles, the state machine enables an output_enable signal on clock cycle 3 to load the result of the multiplication (signal_c) into an output register (register_c).



The design frequency goal is 50 MHz (20 ns) and the multiply function takes 35 ns, but it is given 2 clock cycles. After optimization, this 35 ns path is normally reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To avoid reporting the paths as timing violations, use the SCOPE window to set 2-cycle constraints (From column) on register_a and register_b, or include the following in the timing constraint file:

```
# Paths from register_a use 2 clock cycles
set_multicycle_path -from register_a 2

# Paths from register_b use 2 clock cycles
set_multicycle_path -from register_b 2
```

Alternatively, you can specify a 2-cycle SCOPE constraint (To column) on register_c, or add the following to the constraint file:

```
# Paths to register_c use 2 clock cycles
set_multicycle_path -to register_c 2
```


False Paths

You use the Delay Paths constraint to specify clock paths that you want the synthesis tool to ignore during timing analysis and assign low (or no) priority during optimization. The equivalent Tcl constraint is described in [set_false_path](#), on page 218.

This section describes the following:

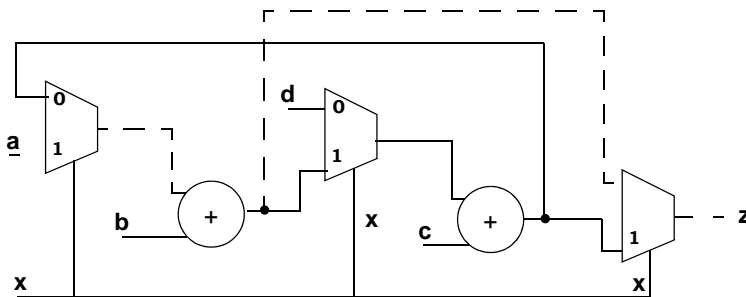
- [Types of False Paths](#), on page 185
- [False Path Constraint Examples](#), on page 186

Types of False Paths

A false path is a path that is not important for timing analysis. There are two types of false paths:

- Architectural false paths

These are false paths that the designer is aware of, like an external reset signal that feeds internal registers but which is synchronized with the clock. The following example shows an architectural false path where the primary input *x* is always 1, but which is not optimized because the software does not optimize away primary inputs.



- Code-introduced false paths

These are false paths that you identify after analyzing the schematic.

False Path Constraint Examples

In this example, the design frequency goal is 50 MHz (20ns) and the path from register_a to register_c is a false path with a large delay of 35 ns. After optimization, this 35 ns path is normally reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To lower the priority of this path during optimization, define it as a false path. You can do this in many ways:

- If all paths from register_a to any register or output pins are not timing-critical, then add a false path constraint to register_a in the SCOPE interface (From), or put the following line in the timing constraint file:

```
#Paths from register_a are ignored  
set_false_path -from {i:register_a}
```

- If all paths to register_c are not timing-critical, then add a false path constraint to register_c in the SCOPE interface (To), or include the following line in the timing constraint file:

```
#Paths to register_c are ignored  
set_false_path -to {i:register_c}
```

- If only the paths between register_a and register_c are not timing-critical, add a From/To constraint to the registers in the SCOPE interface (From and To), or include the following line in the timing constraint file:

```
#Paths to register_c are ignored  
set_false_path -from {i:register_a} -to {i:register_c}
```

Specifying From, To, and Through Points

The following section describes from, to, and through points for timing exceptions specified by the multicycle paths, false paths, and max delay paths constraints.

- [Timing Exceptions Object Types, on page 187](#)
- [From/To Points, on page 187](#)
- [Through Points, on page 189](#)
- [Product of Sums Interface, on page 190](#)
- [Clocks as From/To Points, on page 192](#)

Timing Exceptions Object Types

Timing exceptions must contain the type of object in the constraint specification. You must explicitly specify an object type, n: for a net, or i: for an instance, in the instance name parameter of all timing exceptions. For example:

```
set_multicycle_path -from {i:inst2.lowreg_output[7]}  
                    -to {i:inst1.DATA0[7]} 2
```

If you use the SCOPE GUI to specify timing exceptions, it automatically attaches the object type qualifier to the object name.

From/To Points

From specifies the starting point for the timing exception. To specifies the ending point for the timing exception. When you specify an object, use the appropriate prefix (see [syn_black_box, on page 50](#)) to avoid confusion. The following table lists the objects that can serve as starting and ending points:

From Points**To Points**

Clocks. See [Clocks as From/To Points, on page 192](#) for more information.

Clocks. See [Clocks as From/To Points, on page 192](#) for more information.

Registers

Registers

Top-level input or bi-directional ports

Top-level output or bi-directional ports

Instantiated library primitive cells (gate cells)

Instantiated library primitive cells (gate cells)

Black box outputs

Black box inputs

You can specify multiple from points in a single exception. This is most common when specifying exceptions that apply to all the bits of a bus. For example, you can specify constraints From A[0:15] to B – in this case, there is an exception, starting at any of the bits of A and ending on B.

Similarly, you can specify multiple to points in a single exception. If you specify both multiple starting points and multiple ending points such as From A[0:15] to B[0:15], there is actually an exception from any start point to any end point. In this case, the exception applies to all $16 * 16 = 256$ combinations of start/end points.

Through Points

Through points are limited to nets, hierarchical ports, and pins of instantiated cells. There are many ways to specify these constraints.

- [Single Point](#)
- [Single List of Points](#)
- [Multiple Through Points](#)
- [Multiple Through Lists](#)

You define these constraints in the appropriate SCOPE panels, or in the POS GUI (see [Product of Sums Interface, on page 190](#)). When a port and net have the same name, preface the name of the through point with n: for nets or t: for hierarchical ports. For example, you can specify n:regs_mem[2] or t:dmux.bdpol. The n: prefix must be specified to identify nets; otherwise, the associated timing constraint will not be applied for valid nets.

Single Point

You can specify a single through point. In this case, the constraint is applied to any path that passes through net regs_mem[2] as follows:

```
set_false_path -through n:regs_mem[2]
set_false_path -through [get_nets {regs_mem[2]}]
```

Single List of Points

If you specify a list of through points, the through option behaves as an OR function and applies to any path that passes through any of the points in the list. In the following example, the constraint is applied to any path through regs_mem[2] OR prgcntr.pc[7] OR dmux.alub[0] with a maximum delay value of 5 ns (-max 5):

```
set_max_delay
-through {t:regs_mem[2] t:prgcntr.pc[7] t:dmux.alub[0]} 5
```

Multiple Through Points

You can specify multiple points for the same constraint by preceding each point with the `-through` option. In the following example, the constraint operates as an AND function and applies to paths through `regs_mem[2]` AND `prgcntr.pc[7]` AND `dmux.alub[0]`:

```
set_max_delay
-through t:regs_mem[2]
-through t:prgcntr.pc[7]
-through t:dmux.alub[0] 5
```

Multiple Through Lists

If you specify multiple `-through` lists, the constraint is applied as an AND/OR function and is applied to the paths through all points in the lists. The following constraint applies to all paths that pass through nets { A_1 or A_2 or... A_n } AND nets { B_1 or B_2 or B_3 }:

```
set_false_path -through {n:A1 n:A2...n:An} -through {n:B1 n:B2 n:B3}
```

In this example,

```
set_multicycle_path
-through {n:net1 n:net2}
-through {n:net3 n:net4} 2
```

all paths that pass through the following nets are constrained at 2 clock cycles:

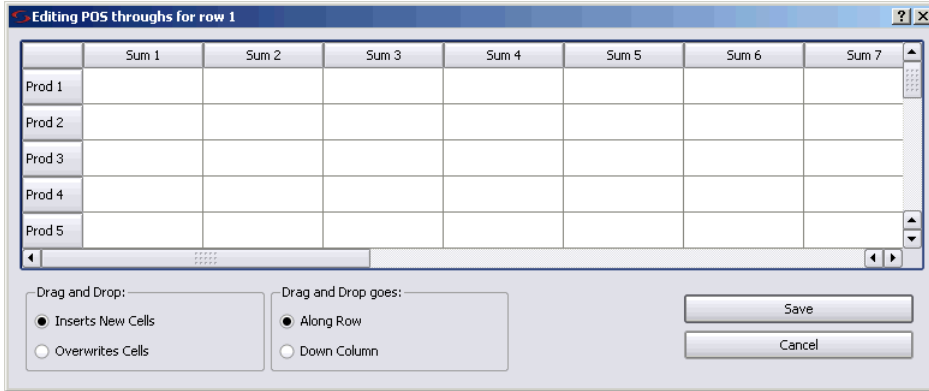
```
net1 AND net3
OR net1 AND net4
OR net2 AND net3
OR net2 AND net4
```

Product of Sums Interface

You can use the SCOPE GUI to format `-through` points for nets with multicycle path, false path, and max delay path constraints in the Product of Sums (POS) interface of the SCOPE editor. You can also manually specify constraints that use the `-through` option. For more information, see [Defining From/To/Through Points for Timing Exceptions](#), on page 132 in the *User Guide*.

The POS interface is accessible by clicking the arrow in a Through column cell in the following SCOPE panels:

- Multi-Cycle Paths
- False Paths
- Delay Paths



Field	Description
Prod 1, 2, etc.	Type the first net name in a cell in a Prod row, or drag the net from a HDL Analyst view into the cell. Repeat this step along the same row, adding other nets in the Sum columns. The nets in each row form an OR list.
Sum 1, 2, etc.	Type the first net name in the first cell in a Sum column, or drag the net from a HDL Analyst view into the cell. Repeat this step down the same Sum column. The nets in each column form an AND list.
Drag and Drop Goes	Along Row - places objects in multiple Sum columns, utilizing only one Prod row. Down Column - places objects in multiple Prod rows, utilizing only one Sum column.
Drag and Drop	Inserts New Cells - New cells are created when dragging and dropping nets. Overwrites Cells - Existing cells are overwritten when dragging and dropping nets.
Save/Cancel	Saves or cancels your session.

Clocks as From/To Points

You can specify clocks as from/to points in your timing exception constraints. Here is the syntax:

```
set_timing_exception -from | -to {c:clock_name[:edge]}
```

where

- *timing_exception* is one of the following constraint types: multicycle path, false path, or max delay
- **c:clock_name:edge** is the name of the clock and clock edge (r or f). If you do not specify a clock edge, by default both edges are used.

See the following sections for details and examples on each timing exception.

Multicycle Path Clock Points

When you specify a clock as a from or to point, the multicycle path constraint applies to all registers clocked by the specified clock.

The following constraint allows two clock periods for all paths from the rising edge of the flip-flops clocked by clk1:

```
set_multicycle_path -from {c:clk1:r} 2
```

You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint allows two clock periods for all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_multicycle_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]) 2
```


False Path Clock Points

When you specify a clock as a from or to point, the false path constraint is set on all registers clocked by the specified clock. False paths are ignored by the timing analyzer. The following constraint disables all paths from the rising edge of the flip-flops clocked by clk1:

```
set_false_path -from {c:clk1:r}
```

You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint disables all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net.

```
set_false_path -to {c:clk1:f} -through (n:MYINST.mybus2[9])
```

Path Delay Clock Points

When you specify a clock as a from or to point for the path delay constraint, the constraint is set on all paths of the registers clocked by the specified clock. This constraint sets a max delay of 2 ns on all paths to the falling edge of the flip-flops clocked by clk1:

```
set_max_delay -to {c:clk1:f} 2
```

You cannot specify a clock as a through point, but you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The next constraint sets a max delay of 0.2 ns on all paths from the rising edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_max_delay -from {c:clk1:r} -through (n:MYINST.mybus2[9]) .2
```

Conflict Resolution for Timing Exceptions

The term *timing exceptions* refers to the false path, max path delay, and multicycle path timing constraints. When the tool encounters conflicts in the way timing exceptions are specified through the constraint file, the software uses a set priority to resolve these conflicts. Conflict resolution is categorized into four levels, meaning that there are four different tiers at which conflicting constraints can occur, with one being the highest. The table below summarizes conflict resolution for constraints. The sections following the table provide more details on how conflicts can occur and examples of how they are resolved.

Conflict Level	Constraint Conflict	Priority	For Details, see ...
1	Different timing exceptions set on the same object.	1 – False Path 2 – Path Delay 3 – Multi-cycle Path	Conflicting Timing Exceptions, on page 195.
2	Timing exceptions of the same constraint type, using different semantics (from/to/through).	1 – From 2 – To 3 – Through	Same Constraint Type with Different Semantics, on page 196.
3	Timing exceptions of the same constraint type using the same semantic, but set on different objects.	1 – Ports/Instances/Pins 2 – Clocks	Same Constraint and Semantics with Different Objects, on page 197.
4	Identical timing constraints, except constraint values differ.	Tightest, or most constricting constraint.	Identical Constraints with Different Values, on page 197.

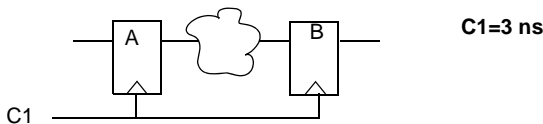
In addition to the four levels of conflict resolution for timing exceptions, there are priorities for the way the tool handles multiple I/O delays set on the same port and implicit and explicit false path constraints. For information on resolving these types of conflicts, see [Priority of Multiple I/O Constraints, on page 166.](#)

Conflicting Timing Exceptions

The first (and highest) level of resolution occurs when timing exceptions—false paths, max path delay, or multicycle path constraints—conflict with each other. The tool follows this priority for applying timing exceptions:

1. False Path
2. Path Delay
3. Multicycle Path

For example:



```
set_false_path -from {c:C1:r}
set_max_delay -from {i:A} -to {i:B} 10
set_multicycle_path -from {i:A} -to {i:B} 2
```

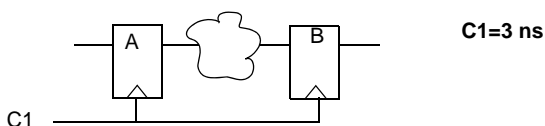
These constraints are conflicting because the path from A to B has three different constraints set on it. When the tool encounters this type of conflict, the false path constraint is honored. Because it has the highest priority of all timing exceptions, `set_false_path` is applied and the other timing exceptions are ignored.

Same Constraint Type with Different Semantics

The second level of resolution occurs when conflicts between timing exceptions that are of the same constraint type, use different semantics (from/to/through). The priority for these constraints is as follows:

1. From
2. To
3. Through

If there are two multicycle constraints set on the same path, one specifying a from point and the other specifying a to point, the constraint using -from takes precedence, as in the following example.



```
set_multicycle_path -from {i:A} 3
set_multicycle_path -to {i:B} 2
```

In this case, the tool uses:

```
set_multicycle_path -from {i:A} 3
```

The other constraint is ignored even though it sets a tighter constraint.

Same Constraint and Semantics with Different Objects

The third level resolves timing exceptions of the same constraint type that use the same semantic, but are set on different objects. The priority for design objects is as follows:

1. Ports/Instances/Pins
2. Clocks

If the same constraints are set on different objects, the tool ignores the constraint set on the clock for that path.

```
set_multicycle_path -from {i:mac1.datax[0]} -start 4
set_multicycle_path -from {c:clk1:r} 2
```

In the example above, the tool uses the first constraint set on the instance and ignores the constraint set on the clock from i:mac1.datax[0], even though the clock constraint is tighter.

For details on how the tool prioritizes multiple I/O delays set on the same port or implicit and explicit false path constraints, see [Priority of Multiple I/O Constraints, on page 166](#).

Identical Constraints with Different Values

Where timing constraints are identical except for the constraint value, the tightest or most constricting constraint takes precedence. In the following example, the tool uses the constraint specifying two clock cycles:

```
set_multicycle_path -from {i:special_regs.trisa[7:0]} 2
set_multicycle_path -from {i:special_regs.trisa[7:0]} 3
```

SCOPE User Interface (Legacy)

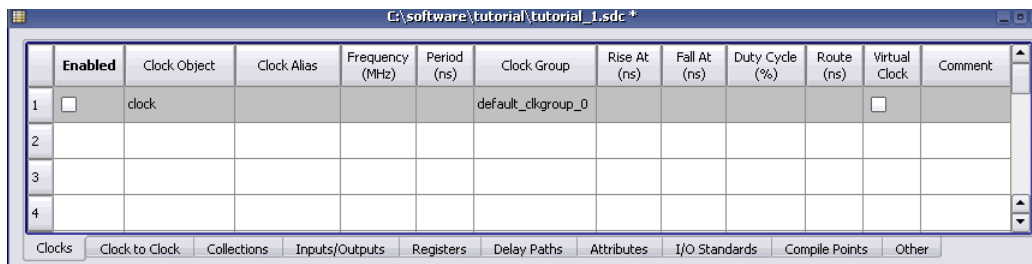
You can use the legacy SCOPE editor for the SDC constraint files created before release version G-2012.09. However, it is recommended that you translate your SDC files to FDC files to enable the latest version of the SCOPE editor and to utilize the enhanced timing constraint handling in the tool. The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

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

```
% sdc2fdc
```

This feature translates all SDC files in your project.

If you want to edit your existing SDC file, to open the legacy SCOPE editor, double-click on your constraint file in the Project view.



The details of the legacy SCOPE interface and constraint syntax are no longer documented here. Refer to the SolvNet article on legacy constraints for details.

CHAPTER 6

Constraint Syntax

The following describe Tcl equivalents for the timing and design constraints you specify in the SCOPE editor or in a constraint file.

- [FPGA Timing Constraints, on page 200](#)
- [Design Constraints, on page 237](#)

FPGA Timing Constraints

The FPGA synthesis tools support FPGA timing constraints for a subset of the clock definition, I/O delay, and timing exception constraints.

For more information about using FPGA timing constraints with your project, see [Using the SCOPE Editor, on page 114](#) in the *User Guide*.

For information on the supported design constraints, see [Design Constraints, on page 237](#).

The remainder of this section describes the constraint file syntax for the following FPGA timing constraints in the FPGA synthesis tools.

- [create_clock](#)
- [create_generated_clock](#)
- [reset_path](#)
- [set_clock_groups](#)
- [set_clock_latency](#)
- [set_clock_route_delay](#)
- [set_clock_uncertainty](#)
- [set_false_path](#)
- [set_input_delay](#)
- [set_max_delay](#)
- [set_multicycle_path](#)
- [set_output_delay](#)
- [set_reg_input_delay](#)
- [set_reg_output_delay](#)

Note: When adding comments for constraints, use standard Tcl syntax conventions. Otherwise, invalid specifications can cause the constraint to be ignored. The (#) comment must begin on a new line or needs to be preceded by a (;), if the comment is on the same line as the constraint. For example:

```
create_clock -period 10 [get_ports CLK]; # comment text
```



```
# comment text
set_clock_groups -asynchronous -group
MMCM_module|clk100_90_MMCM_derived_clock_CLKIN1
```

create_clock

Creates a clock object and defines its waveform in the current design.

Syntax

The supported syntax for the `create_clock` constraint is:

```
create_clock
  -name clockName [-add] {objectList} |
    -name clockName [-add] [{objectList}] |
    [-name clockName [-add]] {objectList}
  -period value
  [-waveform {riseValue fallValue}]
  [-disable]
  [-comment commentString]
```

Arguments

-name <i>clockName</i>	Specifies the name for the clock being created, enclosed in quotation marks or curly braces. If this option is not used, the clock gets the name of the first clock source specified in the <i>objectList</i> option. If you do not specify the <i>objectList</i> option, you must use the -name option, which creates a virtual clock not associated with a port, pin, or net. You can use both the -name and <i>objectList</i> options to give the clock a more descriptive name than the first source pin, port, or net. If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you specify this option, you must also use the -name option.
-period <i>value</i>	Specifies the clock period in nanoseconds. This is the minimum time over which the clock waveform repeats. The <i>value</i> type must be greater than zero.

-waveform <i>riseValue</i> <i>fallValue</i>	Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise followed by fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of <i>periodValue/2</i> .
objectList	Clocks can be defined on the following objects: pins, ports, and nets. The FPGA synthesis tools support nets and instances, where instances have only one output (for example, BUFGs).
-disable	Disables the constraint.
-comment <i>textString</i>	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Examples

Refer to the following examples.

Example 1

A clock named `clk_in1` is created for port `clk_in1` that uses a period of 10 with rising edge of 0 and falling edge of 5.

```
create_clock -name {clk_in1} -period 10 [get_ports {clk_in1}]
```

Example 2

A clock named `clk` is created for port `clk_in` that uses a period of 10.0 with rising edge of 5.0 and falling edge of 9.5.

```
create_clock -name {clk} -period 10 -waveform {5.0 9.5}
[get_ports {clk_in}]
```

Example 3

A virtual clock named `CLK` is created that uses a period of 12 with a rising edge of 0.0 and falling edge of 6.0.

```
create_clock -name {CLK} -period 12
```

create_generated_clock

Creates a generated clock object.

Syntax

The supported syntax for the `create_generated_clock` constraint is:

```
create_generated_clock
  -name clockName [-add] | {clockObject}
  -source masterPinName
  [-master_clock clockName]
  [-divide_by integer | -multiply_by integer [-duty_cycle value]]
  [-invert]
  [-edges {edgeList}]
  [-edge_shift {edgeShiftList}]
  [-combinational]
  [-disable]
  [-comment commentString]
```

Arguments

-name <i>clockName</i>	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the <code>-source</code> option (<i>clockObject</i>). If you specify the <code>-add</code> option, you must use the <code>-name</code> option and the clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also use the <code>-name</code> and <code>-master_clock</code> options.
<i>clockObject</i>	The first clock source specified in the <code>-source</code> option in the absence of <i>clockName</i> . Clocks can be defined on pins, ports, and nets. The FPGA synthesis tools support nets and instances, where instances have only one output (for example, BUFs).
-source <i>masterPinName</i>	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.

-master_clock <i>clockName</i>	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.
-divide_by <i>integer</i>	Specifies the frequency division factor. If the <i>divideFactor</i> value is 2, the generated clock period is twice as long as the master clock period.
-multiply_by <i>integer</i>	Specifies the frequency multiplication factor. If the <i>multiplyFactor</i> value is 3, the generated clock period is one-third as long as the master clock period.
-duty_cycle <i>percent</i>	Specifies the duty cycle, as a percentage, if frequency multiplication is used. Duty cycle is the high pulse width. Note: This option is valid only when used with the -multiply_by option.
-invert	Inverts the generated clock signal (in the case of frequency multiplication and division).
-edges <i>edgeList</i>	Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be set to 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.
-edge_shift <i>edgeShiftList</i>	Specifies a list of floating point numbers that represents the amount of shift, in nanoseconds, that the specified edges are to undergo to yield the final generated clock waveform. The number of edge shifts specified must be equal to the number of edges specified. The values can be positive or negative; positive indicating a shift later in time, while negative indicates a shift earlier in time. For example, 1 indicates that the corresponding edge is to be shifted by one library time unit.
-combinational	The source latency paths for this type of generated clock only includes the logic where the master clock propagates. The source latency paths do not flow through sequential element clock pins, transparent latch data pins, or source pins of other generated clocks.
-disable	Disables the constraint.
-comment <i>textString</i>	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Examples

Refer to the following examples.

Example 1

A generated clock is created whose edges are 1, 3, and 5 of the master clock source. If the master clock period is 30 and the master waveform is {24 36}, then the generated clock period becomes 60 with waveform {24 54}.

```
create_generated_clock -name {genclk} -source  
    [get_ports {clk_in1}] [get_nets {dut.clk_out2}] -edges {1 3 5}
```

Example 2

This example shows the generated clock from the previous example with each derived edge shifted by 1 time unit. If the master clock period is 30 and the master waveform is {24 36}, then the generated clock period becomes 60 with waveform {25 55}.

```
create_generated_clock -name {genclk}  
    -source [get_ports {clk_in1}] [get_nets {dut.clk_out2}]  
    -edges {1 3 5} -edge_shift {1 1 1}
```

Example 3

A generated clock is created, which is $2/5^{\text{th}}$ of the source clock defined at port clk_in1.

```
create_generated_clock -name {genclk}  
    -source [get_ports {clk_in1}] [get_nets {dut.clk_out2}]  
    -edges {1 1 2} -edge_shift {0 0.8 -0.4}
```

reset_path

Resets the specified paths to single-cycle timing.

Syntax

The supported syntax for the reset_path constraint is:

```
reset_path [-setup]
           [-from {objectList}]
           [-through {objectList} [-through {objectList} ...] ]
           [-to {objectList}]
           [-disable]
           [-comment commentString]
```

Arguments

-setup	Specifies that setup checking (maximum delay) is reset to single-cycle behavior.
-from	<p>Specifies the names of objects to use to find path start points. The -from <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Clocks • Registers • Top-level input or bi-directional ports) • Black box outputs • Sequential cell clock pins <p>When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points</p>

-through	<p>Specifies the intermediate points for the timing exception. The -through <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Combinational nets • Hierarchical ports • Pins on instantiated cells <p>By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in <i>objectList</i>. If more than one object is included, the objects must be enclosed either in quotation marks (") or in braces ({}). If you specify the -through option multiple times, reset_path applies to the paths that pass through a member of each <i>objectList</i>. If you use the -through option in combination with the -from or -to options, reset_path applies only if the -from or -to and the -through conditions are satisfied.</p>
-to	<p>Specifies the names of objects to use to find path end points. The -to <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins <p>If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.</p>
-disable	Disables the constraint.
-comment <i>textString</i>	<p>Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.</p>

set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design. Clocks created with `create_clock` are considered synchronous as long as no `set_clock_groups` constraints specify otherwise. Paths between asynchronous clocks are not considered for timing analysis.

Clock grouping in the FPGA synthesis environment is inclusionary or exclusionary. For example, `clk2` and `clk3` can each be related to `clk1` without being related to each other.

Syntax

```
set_clock_groups
  -asynchronous | -physically_exclusive | -logically_exclusive
  [-name clockGroupName]
  -group {clockList} [-group {clockList} ... ]
  -derive
  [-disable]
  [-comment commentString]
```

Arguments

-asynchronous	Specifies that the clock groups are asynchronous to each other (the FPGA synthesis tools assume all clock groups are synchronous). Two clocks are asynchronous with respect to each other if they have no phase relationship at all.
-physically_exclusive	Specifies that the clock groups are physically exclusive to each other. An example is multiple clocks that are defined on the same source pin. The FPGA synthesis tools accept this option, but treats it as -asynchronous.
-logically_exclusive	Specifies that the clock groups are logically exclusive to each other. An example is multiple clocks that are selected by a multiplexer, but might have coupling with each other in the design. The FPGA synthesis tools accept this option, but treats it as -asynchronous.

-name {clockGroupName}	Specifies a unique name for a clock grouping. This option allows you to easily identify specified clock groups, which are exclusive or asynchronous with all other clock groups in the design.
-group {clockList}	<p>Specifies a space-separated list of clocks in <i>{clockList}</i> that are asynchronous to all other clocks in the design, or asynchronous to the clocks specified in other -group arguments in the same command.</p> <p>If you specify only one group, the clocks in that group are exclusive or asynchronous with all other clocks in the design. Whenever a new clock is created, it is automatically included in the default “other” group that includes all the other clocks in the design.</p> <p>If you specify -group multiple times in a single command execution, the listed clocks are only asynchronous with the clocks in the other groups specified in the same command. You can include a clock in only one group in a single command execution. To include a clock in multiple groups, use multiple <code>set_clock_groups</code> commands.</p> <p>Do not use commas between clock names in the list. See -group Option, on page 211.</p>
-derive	Specifies that generated and derived clocks inherit the clock group of the parent clock. By default, a generated clock and its master clock are not in the same group when the exclusive or asynchronous clock groups are defined. The -derive option lets you override this behavior and allow generated or derived clocks to inherit the clock group of their parent source clock.
-disable	Disables the constraint.
-comment <i>textString</i>	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Restrictions

Be aware of the restrictions for the following `set_clock_groups` options:

-group Option

Do not insert commas between clock names when you use the -group option, because the tool treats the comma as part of the clock name. This is true for all constraints that contain lists. This means that if you specify the following constraint, the tool generates a warning that it cannot find clk1,:

```
set_clock_groups -asynchronous -group {clk1, clk2}
```

Examples

The following examples illustrate how to use this constraint.

Example 1

This `set_clock_groups` constraint specifies that `clk4` is asynchronous to all other clocks in the design.

```
set_clock_groups -asynchronous -group {clk4}
```

Example 2

This `set_clock_groups` constraint specifies that clock `clk1`, `clk2`, and `clk3` are asynchronous to all other clocks in the design. If a new clock called `clkx` is added to the design, `clk1`, `clk2`, and `clk3` are asynchronous to it too.

```
set_clock_groups -asynchronous -group {clk1 clk2 clk3}
```

Example 3

The following `set_clock_groups` constraint has multiple -group arguments, and specifies that `clk1` and `clk2` are asynchronous to `clk3` and `clk4`.

```
set_clock_groups -asynchronous -group {clk1 clk2}  
-group {clk3 clk4}
```

Example 4

The following `set_clock_groups` constraint specifies that `clk1` and `clk2` which were synchronous when defined with the `create_clock` command, are now asynchronous.

```
create_clock [get_ports {c1}] -name clk1 -period 10
create_clock [get_ports {c2}] -name clk2 -period 16
create_clock [get_ports {c3}] -name clk3 -period 5
set_clock_groups -asynchronous -group [get_clocks {clk1}]
                    -group [get_clocks {clk2}]
```

The following constructs are equivalent:

```
set_clock_groups -asynchronous -group [get_clocks {clk1}]
set_clock_groups -asynchronous -group {clk1}
```

Example 5

The following constraint specifies that test|clkout0_derived_clock_CLKIN1 and test|clkout1_derived_clock_CLKIN1 are asynchronous to all other clocks in the design:

```
set_clock_groups -asynchronous -group [get_clocks {*clkout*}]
```

Example 6

This example defines the clock on the u1.clkout0 net is asynchronous to all other clocks in the design:

```
set_clock_groups -asynchronous -group [get_clocks -of_objects
{n:u1.clkout0}]
```

set_clock_latency

Specifies clock network latency.

Syntax

The supported syntax for the `set_clock_latency` constraint is:

```
set_clock_latency
  -source
    [-clock {clockList}]
    delayValue
    {objectList}
    [-disable]
```

Arguments

-source	Indicates that the specified delay is applied to the clock source latency.
-clock clockList	Indicates that the specified delay is applied with respect to the specified clocks. By default, the specified delay is applied to all specified objects.
<i>delayValue</i>	Specifies the clock latency value.
<i>objectList</i>	Specifies the input ports for which clock latency is to be set

Description

In the FPGA synthesis tools, the `set_clock_latency` constraint accepts both clock objects and clock aliases. Applying a `set_clock_latency` constraint on a port can be used to model the off-chip clock delays in a multi-chip environment. Clock latency is forward annotated in the top-level constraint file as part of the time budgeting that takes place in the Certify/HAPS flow. The annotated values represent the arrival times for clocks on specific ports of any particular FPGA in a HAPS design.

In the above syntax, *objectList* references either input ports with defined clocks or clock aliases defined on the input ports. When more than one clock is defined for an input port, the `-clock` option can be used to apply different latency values to each alias.

Restrictions

The following limitations are present in the FPGA synthesis environment:

- Clock latency can only be applied to clocks defined on input ports.
- The `set_clock_latency` constraint is only used for source latency.
- The constraint only applies to port clock objects.
- Latency on clocks defined with `create_generated_clock` is not supported.

set_clock_route_delay

Translates the -route option for the legacy define_clock constraint.

Syntax

The supported syntax for the set_clock_route_delay constraint is:

```
set_clock_route_delay {clockAliasList} {delayValue}
```

Arguments

<i>clockAliasList</i>	Lists the clock aliases to include the route delay.
<i>delayValue</i>	Specifies the route delay value.

Description

The sdc2fdc translator performs a translation of the -route option for the legacy define_clock constraint and places a set_clock_route_delay constraint in the *_translated.fdc file using the following format:

```
set_clock_route_delay [get_clocks {clk_alias_1 clk_alias_2 ...}]  
    {delay_in_ns}
```

set_clock_uncertainty

Specifies the uncertainty (skew) of the specified clock networks.

Syntax

The supported syntax for the `set_clock_uncertainty` constraint is:

```
set_clock_uncertainty
  {objectList}
  -from fromClock | -rise_from riseFromClock | -fall_from fallFromClock
  -to toClock | -rise_to riseToClock | -fall_to fallToClock
  value
```

Arguments

<i>objectList</i>	Specifies the clocks for simple uncertainty. The uncertainty is applied to the capturing latches clocked by one of the specified clocks. You must specify either this argument or a clock pair with the <code>-from</code> / <code>-rise_from</code> / <code>-fall_from</code> and <code>-to</code> / <code>-rise_to</code> / <code>-fall_to</code> options; you cannot specify both an object list and a clock pair.
-from <i>fromClock</i>	Specifies the source clocks for interclock uncertainty. You can use only one of the <code>-from</code> , <code>-rise_from</code> , and <code>-fall_from</code> options and you must specify a destination clock with one of the <code>-to</code> , <code>-rise_to</code> , and <code>-fall_to</code> options.
-rise_from <i>riseFromClock</i>	Specifies that the uncertainty applies only to the rising edge of the source clock. You can use only one of the <code>-from</code> , <code>-rise_from</code> , and <code>-fall_from</code> options and you must specify a destination clock with one of the <code>-to</code> , <code>-rise_to</code> , and <code>-fall_to</code> options.
-fall_from <i>fallFromClock</i>	Specifies that the uncertainty applies only to the falling edge of the source clock. You can use only one of the <code>-from</code> , <code>-rise_from</code> , and <code>-fall_from</code> options and you must specify a destination clock with one of the <code>-to</code> , <code>-rise_to</code> , and <code>-fall_to</code> options.
-to <i>toClock</i>	Specifies the destination clocks for interclock uncertainty. You can use only one of the <code>-to</code> , <code>-rise_to</code> , and <code>-fall_to</code> options and you must specify a source clock with one of the <code>-from</code> , <code>-rise_from</code> , and <code>-fall_from</code> options.
-rise_to <i>riseToClock</i>	Specifies that the uncertainty applies only to the rising edge of the destination clock. You can use only one of the <code>-to</code> , <code>-rise_to</code> , and <code>-fall_to</code> options and you must specify a source clock with one of the <code>-from</code> , <code>-rise_from</code> , and <code>-fall_from</code> options.

-fall_to <i>fallToClock</i>	Specifies that the uncertainty applies only to the falling edge of the destination clock. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.
<i>value</i>	Specifies a floating-point number that indicates the uncertainty value. Only positive uncertainty numbers are acceptable.

Examples

Refer to the following examples.

Example 1

All paths to registers clocked by clk are specified with setup uncertainty of 0.4 in the following example:

```
set_clock_uncertainty 0.4 -setup [get_clocks clk]
```

Example 2

For this example, interclock uncertainties are specified between clock clk and clk2:

```
set_clock_uncertainty -from [get_clocks clk] -to
[get_clocks clk2] 0.2

set_clock_uncertainty -from [get_clocks clk2] -to
[get_clocks clk] 0.1
```

Example 3

For this example, interclock uncertainties are specified between clock clk and clk2 with specific edges:

```
set_clock_uncertainty -rise_from [get_clocks clk2] -to
[get_clocks clk] 0.5

set_clock_uncertainty -rise_from [get_clocks clk2] -rise_to
[get_clocks clk] 0.1

set_clock_uncertainty -from [get_clocks clk2] -fall_to
[get_clocks clk] 0.1
```

set_false_path

Removes timing constraints from particular paths.

Syntax

The supported syntax for the `set_false_path` constraint is:

```
set_false_path  
  [-setup]  
  [-from {objectList}]  
  [-through {objectList} [-through {objectList} ...] ]  
  [-to {objectList}]  
  [-disable]  
  [-comment commentString]
```

Arguments

-setup	Specifies that setup checking (maximum delay) is reset to single-cycle behavior.
-from	<p>Specifies the names of objects to use to find path start points. The <code>-from</code> <i>objectList</i> includes:</p> <ul style="list-style-type: none">• Clocks• Registers• Top-level input or bi-directional ports• Black box outputs• Sequential cell clock pins• When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points.

-through	<p>Specifies the intermediate points for the timing exception. The -through <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Combinational nets • Hierarchical ports • Pins on instantiated cells <p>By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in <i>objectList</i>. If more than one object is included, the objects must be enclosed either in quotation marks (") or in braces ({}). If you specify the -through option multiple times, <i>set_path</i> applies to the paths that pass through a member of each <i>objectList</i>. If you use the -through option in combination with the -from or -to options, <i>set_false_path</i> applies only if the -from or -to and the -through conditions are satisfied.</p>
-to	<p>Specifies the names of objects to use to find path end points. The -to <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins <p>If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.</p>
-disable	Disables the constraint.
-comment <i>textString</i>	<p>Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.</p>

Examples

Refer to the following examples.

Example 1

All the paths from the sequential cell output pins and clock pins in module *gen_sub[*].u_sub* with names matching *out** (i.e. registers *out1* and *out2* in modules *gen_sub[0].u_sub*, *gen_sub[1].u_sub*, and *gen_sub[2].u_sub*) are set as false paths.

```
set_false_path -from [get_pins {gen_sub\[*\]\.u_sub.out*.*}]
```

Note: Only sequential clock pins are valid pins that can be used as start points for a timing path. Note that hierarchical module pins are not valid as starting points for a timing path.

Example 2

All the paths from the sequential cells in module `gen_sub\[*\].u_sub` with names matching `out*` (i.e. registers `out1` and `out2` in modules `gen_sub\[0\].u_sub`, `gen_sub\[1\].u_sub`, and `gen_sub\[2\].u_sub`) are set as false paths.

```
set_false_path -from [get_cells {gen_sub\[*\]\.u_sub.out*}]
```

Example 3

All paths from top-level input ports with names `in*` are set as false paths.

```
set_false_path -from [get_ports {in*}]
```

Note: Only top-level ports are valid port based start points for timing paths. Do not use the `get_ports` command to reference hierarchical module pins.

Example 4

All paths with end points clocked by clock `clka` are set as false paths.

```
set_false_path -to [get_clocks {clka}]
```

set_input_delay

Sets input delay on pins or input ports relative to a clock signal.

Syntax

The supported syntax for the `set_input_delay` constraint is:

```
set_input_delay
  [-clock clockName [-clock_fall]]
  [-rise|-fall]
  [-min|-max]
  [-add_delay]
  delayValue
  {portPinList}
  [-disable]
  [-comment commentString]
```

Argument

-clock <i>clockName</i>	Specifies the clock to which the specified delay is related. If <code>-clock_fall</code> is used, <code>-clock <i>clockName</i></code> must be specified. If <code>-clock</code> is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.
-clock_fall	Specifies that the delay is relative to the falling edge of the clock. The default is the rising edge.
-rise	Specifies that <i>delayValue</i> refers to a rising transition on the specified ports of the current design. If neither <code>-rise</code> nor <code>-fall</code> is specified, rising and falling delays are assumed to be equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the <code>-rise</code> option is preserved and forward annotated to the place-and-route tool.

-fall	<p>Specifies that <i>delayValue</i> refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed equal.</p> <p>Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.</p>
-min	<p>Specifies that <i>delayValue</i> refers to the shortest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal.</p> <p>Note: The synthesis tool does not optimize for hold time violations and only reports -min delay values in the <code>synlog/topLevel_fpga_mapper.srr_Min</code> timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.</p>
-max	<p>Specifies that <i>delayValue</i> refers to the longest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal.</p> <p>Note: The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.</p>
-add_delay	<p>Specifies if delay information is to be added to the existing input delay or if is to be overwritten. The -add_delay option enables you to capture information about multiple paths leading to an input port that are relative to different clocks or clock edges.</p>
-disable	<p>Disables the constraint.</p>
-comment <i>textString</i>	<p>Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.</p>
<i>delayValue</i>	<p>Specifies the path delay. The <i>delayValue</i> must be in units consistent with the technology library used during optimization. The <i>delayValue</i> represents the amount of time the signal is available after a clock edge. This represents a combinational path delay from the clock pin of a register.</p>
<i>portPinList</i>	<p>Specifies a list of input port names in the current design to which <i>delayValue</i> is assigned. If more than one object is specified, the objects are enclosed in quotes (") or in braces ({}).</p>

set_max_delay

Specifies a maximum delay target for paths in the current design.

Syntax

The supported syntax for the `set_max_delay` constraint is:

```
set_max_delay  
  [-from {objectList}]  
  [-through {objectList} [-through {objectList} ...] ]  
  [-to {objectList}]  
  delayValue  
  [-disable]  
  [-comment commentString]
```

Arguments

- from** Specifies the names of objects to use to find path start points. The `-from objectList` includes:
- Clocks
 - Registers
 - Top-level input or bi-directional ports
 - Black box outputs
 - Sequential cell clock pins
- When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. All paths from these start points to the end points in the `-from objectList` are constrained to `delayValue`. If a `-to objectList` is not specified, all paths from the `-from objectList` are affected. If you include more than one object, you must enclose the objects in quotation marks (") or braces ({}).
-

-through	<p>Specifies the intermediate points for the timing exception. The -through <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Combinational nets • Hierarchical ports • Pins on instantiated cells <p>By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in <i>objectList</i>. The max delay value applies only to paths that pass through one of the points in the -through <i>objectList</i>. If more than one object is included, the objects must be enclosed either in quotation marks (") or in braces ({}). If you specify the -through option multiple times, set_max_delay applies to the paths that pass through a member of each <i>objectList</i>. If you use the -through option in combination with the -from or -to options, set_max_delay applies only if the -from or -to and the -through conditions are satisfied.</p>
-to	<p>Specifies the names of objects to use to find path end points. The -to <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins <p>If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. All paths to the end points in the -to <i>objectList</i> are constrained to <i>delayValue</i>. If a -from <i>objectList</i> is not specified, all paths to the -to <i>objectList</i> are affected. If you include more than one object, you must enclose the objects in quotation marks (") or braces ({}).</p>
-disable	Disables the constraint.
-comment <i>textString</i>	<p>Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.</p>

delayValue

Specifies the value of the desired maximum delay for paths between start and end points. You must express *delayValue* in the same units as the technology library used during optimization. If a path start point is on a sequential device, clock skew is included in the computed delay. If a path start point has an input delay specified, that delay value is added to the path delay. If a path end point is on a sequential device, clock skew and library setup time are included in the computed delay. If the end point has an output delay specified, that delay is added into the path delay.

set_multicycle_path

Modifies the single-cycle timing relationship of a constrained path.

Syntax

The supported syntax for the `set_multicycle_path` constraint is:

```
set_multicycle_path  
  [-start | -end]  
  [-from {objectList}]  
  [-through {objectList} [-through {objectList} ...] ]  
  [-to {objectList}]  
  pathMultiplier  
  [-disable]  
  [-comment commentString]
```

Arguments

-start | -end Specifies if the multi-cycle information is relative to the period of either the start clock or the end clock. These options are only needed for multi-frequency designs; otherwise start and end are equivalent. The start clock is the clock source related to the register or primary input at the path start point. The end clock is the clock source related to the register or primary output at the path endpoint. The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with -end moves the relation forward one cycle of the end clock. A setup multiplier of 2 with -start moves the relation back one cycle of the start clock. A hold multiplier of 1 with -start moves the relation forward one cycle of the start clock. A hold multiplier of 1 with -end moves the relation back one cycle of the end clock. If you do not provide -start or -end, -end is assumed.

-from	<p>Specifies the names of objects to use to find path start points. The -from <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Clocks • Registers • Top-level input or bi-directional ports • Black box outputs • Sequential cell clock pins <p>When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. If a -to <i>objectList</i> is not specified, all paths from the -from <i>objectList</i> are affected. If you include more than one object, you must enclose the objects in quotation marks (") or braces ({}).</p>
-through	<p>Specifies the intermediate points for the timing exception. The -through <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Combinational nets • Hierarchical ports • Pins on instantiated cells <p>The multi-cycle values apply only to paths that pass through one of the points in the -through <i>objectList</i>. If more than one object is included, the objects must be enclosed either in double quotation marks (") or in braces ({}). If you specify the -through option multiple times, set_multicycle_delay applies to the paths that pass through a member of each <i>objectList</i>. If the -through option is used in combination with the -from or -to options, the multi-cycle values apply only if the -from or -to conditions and the -through conditions are satisfied.</p>
-to	<p>Specifies the names of objects to use to find path end points. The -to <i>objectList</i> includes:</p> <ul style="list-style-type: none"> • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins <p>If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. If a -from <i>objectList</i> is not specified, all paths to the -to <i>objectList</i> are affected. If you include more than one object, you must enclose the objects in quotation marks (") or braces ({}).</p>
-disable	<p>Disables the constraint.</p>

-comment <i>textString</i>	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
<i>pathMultiplier</i>	Specifies the number of cycles that the data path must have for setup or hold relative to the start point or end point clock before data is required at the end point. When used with -setup, this value is applied to setup path calculations. When used with -hold, this value is applied to hold path calculations. If neither -hold nor -setup are specified, <i>pathMultiplier</i> is used for setup and (<i>pathMultiplier</i> - 1) is used for hold. Changing the <i>pathMultiplier</i> for setup also affects the hold check.

Examples

Refer to the following examples.

Example 1

All the paths from the sequential cell output pins and clock pins in module `gen_sub[*].u_sub` with names matching `out*` (i.e. registers `out1` and `out2` in modules `gen_sub[0].u_sub`, `gen_sub[1].u_sub`, and `gen_sub[2].u_sub`) provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -from [get_pins{gen_sub\[*\]\.u_sub.out*.*}] 2
```

Note: Only sequential clock pins are pins that can be used as valid start points for a timing path. Note that hierarchical module pins cannot be used as starting points for a timing path.

Example 2

All the paths from the sequential cells in module `gen_sub[*].u_sub` with names matching `out*` (i.e. registers `out1` and `out2` in modules `gen_sub[0].u_sub`, `gen_sub[1].u_sub`, and `gen_sub[2].u_sub`) support the timing cycle set to 2.

```
set_multicycle_path -from [get_cells {gen_sub\[*\]\.u_sub.out*}] 2
```

Example 3

All paths from top-level input ports with names in* provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -from [get_ports {in*}] 2
```

Note: Only top-level ports are valid port based start points for timing paths. Do not use the `get_ports` command to reference hierarchical module pins.

Example 4

All paths with end points clocked by clock `clka` provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -to [get_clocks {clka}] 2
```

set_output_delay

Sets output delay on pins or output ports relative to a clock signal.

Syntax

The supported syntax for the `set_output_delay` constraint is:

```
set_output_delay
  [-clock clockName [-clock_fall]]
  [-rise|[-fall]]
  [-min|-max]
  [-add_delay]
  delayValue
  {portPinList}
  [-disable]
  [-comment commentString]
```

Arguments

-clock <i>clockName</i>	Specifies the clock to which the specified delay is related. If <code>-clock_fall</code> is used, <code>-clock <i>clockName</i></code> must be specified. If <code>-clock</code> is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.
-clock_fall	Specifies that the delay is relative to the falling edge of the clock. If <code>-clock</code> is specified, the default is the rising edge.
-rise	Specifies that <i>delayValue</i> refers to a rising transition on the specified ports of the current design. If neither <code>-rise</code> nor <code>-fall</code> is specified, rising and falling delays are assumed to be equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the <code>-rise</code> option is preserved and forward annotated to the place-and-route tool.

-fall	<p>Specifies that <i>delayValue</i> refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed equal.</p> <p>Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.</p>
-min	<p>Specifies that <i>delayValue</i> refers to the shortest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal.</p> <p>Note: The synthesis tool does not optimize for hold time violations and only reports -min delay values in the <code>synlog/topLevel_fpga_mapper.srr_Min</code> timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.</p>
-max	<p>Specifies that <i>delayValue</i> refers to the longest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal.</p> <p>Note: The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.</p>
-add_delay	<p>Specifies whether to add delay information to the existing output delay or to overwrite. The -add_delay option enables you to capture information about multiple paths leading to an output port that are relative to different clocks or clock edges.</p>
-disable	<p>Disables the constraint.</p>
-comment <i>textString</i>	<p>Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.</p>
<i>delayValue</i>	<p>Specifies the path delay. The <i>delayValue</i> must be in units consistent with the technology library used during optimization. The <i>delayValue</i> represents the amount of time that the signal is required before a clock edge. For maximum output delay, this usually represents a combinational path delay to a register plus the library setup time of that register. For minimum output delay, this value is usually the shortest path delay to a register minus the library hold time</p>

portPinList

A list of output port names in the current design to which *delayValue* is assigned. If more than one object is specified, the objects are enclosed in double quotation marks (") or in braces {}.

set_reg_input_delay

Speeds up paths feeding a register by a given number of nanoseconds.

Syntax

```
set_reg_input_delay {registerName} [-route ns] [-disable] [-comment textString]
```

Arguments

<i>registerName</i>	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The `set_reg_input_delay` timing constraint speeds up paths feeding a register by a given number of nanoseconds. The Synopsys FPGA synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with `create_clock`). Use this constraint to speed up the paths feeding a register. For information about the equivalent SCOPE spreadsheet interface, see [Registers, on page 167](#).

Use this constraint instead of the legacy constraint, `define_reg_input_delay`.

set_reg_output_delay

Speeds up paths coming from a register by a given number of nanoseconds.

Syntax

```
set_reg_output_delay {registerName} [-route ns] [-disable] [-comment textString]
```

Arguments

<i>registerName</i>	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths from the register.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The `set_reg_output_delay` constraint speeds up paths coming from a register by a given number of nanoseconds. The synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with `create_clock`). Use this constraint to speed up the paths coming from a register. For information about the equivalent SCOPE spreadsheet interface, see [Registers, on page 167](#).

Use this constraint instead of the legacy constraint, `define_reg_output_delay`.

Naming Rule Syntax Commands

The FPGA synthesis environment uses a set of naming conventions for design objects in the RTL when your project contains constraint files. The following naming rule commands are added to the constraint file to change the expected default values. These commands must appear at the beginning of

the constraint file before any other constraints. Similarly, when multiple constraint files are included in the project, the naming rule commands must be in the first constraint file read.

set_hierarchy_separator Command

The `set_hierarchy_separator` command redefines the hierarchy separator character (the default separator character is the period in the FPGA synthesis environment). For example, the following command changes the separator character to a forward slash:

```
set_hierarchy_separator {/}
```

Embedded Tcl commands, such as `get_pins` must be enclosed in brackets `[]` for the software to execute the command. Also, the curly brackets `{ }` are required when object names include the escape `(\)` character or square brackets. For example, the following syntax is honored by the tool:

```
set_hierarchy_separator {/}  
create_clock -name {clk1} [get_pins  
{pdp_c/ib_phy_c/port_g\1\phy_c/c7_g\gtxe2_common_0_i/GTREFCLK[0]}]  
-period {10}
```

set_rtl_ff_names Command

The `set_rtl_ff_names` command controls the stripping of register suffixes in the object strings of delay-path constraints (for example, `set_false_path`, `set_multicycle_path`). Generally, it is only necessary to change this value from its default when constraints that target ASIC designs are being imported from the Design Compiler (in the Design Compiler, inferred registers are given a `_reg` suffix during the elaboration phase; constraints targeting these registers must include this suffix). When importing constraints from the Design Compiler, include the following command to change the value of this naming rule to `{_reg}` to automatically recognize the added suffix.

```
set_rtl_ff_names {_reg}
```

For example, using the above value allows the DC exception

```
set_false_path -to [get_cells {register_bus_reg[0]}]
```

to apply to the following object without having to manually modify the constraint:

```
[get_cells {register_bus[0]}]
```

bus_naming_style Command

The `bus_naming_style` command redefines the format for identifying bits of a bus (by default, individual bits of a bus are identified by the bus name followed by the bus bit enclosed in square brackets). For example, the following command changes the bus-bit identification from the default *busName[busBit]* format to the *busName_busBit* format:

```
bus_naming_style {%s_%d}
```

bus_dimension_separator_style Command

The `bus_dimension_separator_style` command redefines the format for identifying multi-dimensional arrays (by default, multidimensional arrays such as row 2, bit 3 of array `ABC[n x m]` are identified as `ABC[2][3]`). For example, the following command changes the bus-dimension separator from individual square bracket sets to an underscore:

```
bus_dimension_separator_style {_}
```

The resulting format for the above example is:

```
ABC[2_3]
```

read_sdc Command

Reads in a script in Synopsys FPGA constraint format. The supported syntax for the `read_sdc` constraint is:

```
read_sdc fileName
```

Design Constraints

This section describes the constraint file syntax for the following non-timing design constraints:

- [define_compile_point](#), on page 238
- [define_current_design](#), on page 239
- [define_io_standard](#), on page 240

define_compile_point

The `define_compile_point` command defines a compile point in a top-level constraint file. You use one `define_compile_point` command for each compile point you define. For the equivalent SCOPE spreadsheet interface, see [Compile Points, on page 173](#). (Compile points are only available for certain technologies.)

This is the syntax:

```
define_compile_point [-disable ] {moduleName}  
    -type {soft|hard|locked|locked, partition} [-comment textString ]
```

-disable Disables a previous compile point definition.

-type Specifies the type of compile point. This can be soft, hard, locked, or locked, partition. See [Compile Point Types, on page 419](#) for more information.

Refer to [Guidelines for Entering and Editing Constraints, on page 129](#) for details about the syntax and prefixes for naming objects.

Here is a syntax example:

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

define_current_design

The `define_current_design` command specifies the module to which the constraints that follow it apply. It must be the first command in a block-level or compile-point constraint file. The specified module becomes the top level for objects defined in this hierarchy and the constraints applied in the respective block-level or compile-point constraint file.

This is the syntax:

```
define_current_design {regionName | libraryName.moduleName}
```

Refer to [Guidelines for Entering and Editing Constraints](#), on page 129 for details about the syntax and prefixes for naming objects.

Here is an example:

```
define_current_design {lib1.prgm_cntr}
```

Objects in all constraints that follow this command relate to `prgm_cntr`.

define_io_standard

Specifies a standard I/O pad type to use for various Microsemi families. See [I/O Standards, on page 171](#) for details of the SCOPE equivalent.

```
define_io_standard [-disable] {p:portName} -delay_type input|output|bidir
  syn_pad_type {IO_standard} [parameter {value}...]
```

In the above syntax:

portName is the name of the input, output, or bidirectional port.

-delay_type identifies the port direction which must be input, output, or bidir.

syn_pad_type is the I/O pad type (I/O standard) to be assigned to *portName*.

parameter is one or more of the parameters defined in the following table. Note that these parameters are device-family dependent.

Parameter	Function
syn_io_termination	The termination type; typical values are pullup and pulldown.
syn_io_drive	The output drive strength; values include low and high or numerical values in mA.
syn_io_dv2	Switch to use a 2x impedance value (DV2).
syn_io_dci	Switch for digitally-controlled impedance (DCI).
syn_io_slew	The slew rate for single-ended output buffers; values include slow and fast or low and high.

Examples:

```
define_io_standard {p:DATA1[7:0]} -delay_type input
  syn_pad_type {LVCMOS_33} syn_io_slew {high}
  syn_io_drive {12} syn_io_termination {pulldown}
```


CHAPTER 7

Input and Result Files

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

- [Input Files, on page 242](#)
- [Libraries, on page 245](#)
- [Output Files, on page 249](#)
- [Log File, on page 254](#)
- [Timing Reports, on page 259](#)
- [Constraint Checking Report, on page 268](#)

Input Files

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

Extension	File	Description
.adc	Analysis Design Constraint	<p>Contains timing constraints to use for stand-alone timing analysis. Constraints in this file are used only for timing analysis and do not change the result files from synthesis. Constraints in the adc file are applied in addition to sdc constraints used during synthesis. Therefore, adc constraints affect timing results only if there are no conflicts with sdc constraints.</p> <p>You can forward annotate adc constraints to your vendor constraint file without rerunning synthesis. See Using Analysis Design Constraints, on page 335 of the <i>User Guide</i> for details.</p>
.fdc	Synopsys FPGA Design Constraint	<p>Create FPGA timing and design constraints with SCOPE. You can run the sdc2fdc utility to translate legacy FPGA timing constraints (SDC) to Synopsys FPGA timing constraints (FDC). For details, see the sdc2fdc, on page 54.</p>
.ini	Configuration and Initialization	<p>Governs the behavior of the synthesis tool. You normally do <i>not</i> need to edit this file. For example, use the HDL Analyst Options dialog box, instead, to customize behavior. See New HDL Analyst Options Command, on page 319.</p> <p>On the Windows 7 platforms, the ini file is in the C:\Users\userName\AppData\Roaming\Synplicity directory</p> <p>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).</p>
.prj	Project	<p>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.</p>
.sdc	Constraint	<p>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.</p>

Extension	File	Description
.sv	Source files (Verilog)	Design source files in SystemVerilog format. The sv source file is added to the Verilog directory in the Project view. For more information about the Verilog and SystemVerilog languages, and the synthesis commands and attributes you can include, see Verilog, on page 244, 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 43 of the User Guide .
.vhd	Source files (VHDL)	Design source files in VHDL format. See VHDL, on page 244 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 43 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 244, 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 43 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 two kinds of libraries you can use:

- Technology-specific libraries that contain I/O pad, macro, or other component descriptions. The *lib* directory lists these kinds of libraries under vendor sub-directories. The libraries are named for the technology family, and in some cases also include a version number for the version of the place-and-route tool with which they are intended to be used.

For information about using vendor-specific libraries to instantiate LPMs, macros, I/O pads, and other components, refer to the appropriate sections in [Chapter 15, *Optimizing for Microsemi Designs*](#) in the *User Guide*.

- The open verification library is automatically included in the FPGA product installation. When using your own open verification library, follow the recommendation described in [Open Verification Library \(Verilog\)](#), on page 246.
- 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 246 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 247](#).

Open Verification Library (Verilog)

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

```
#in bash
export disable_synovl=1

#in csh
setenv disable_synovl 1
```

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

The Generic Technology Library

The synthesis software includes this Verilog library for generic components under the *installDirectory/lib/generic_technology* directory. Currently, the library is only available in Verilog format. The library consists of technology-independent common logic elements, which help the designer to develop technology-independent parts. The library models extract the functionality of the component, but not its implementation. During synthesis, the mappers implement these generic components in implementations that are appropriate to the technology being used.

To use components from this directory, add the library to the project by doing either of the following:

- Add `add_file -verilog "$LIB/generic_technology/gtech.v` to your `prj` file or type it in the Tcl window.

- In the tool window, click the Add file button, navigate to the *installDirectory/lib/generic_technology* directory and select the *gtech.v* file.

When you synthesize the design, the tool uses components from this library.

You cannot use the generic technology library together with other generic libraries, as this could result in a conflict. If you have your own GTECH library that you intend to use, do not use the generic technology library.

ASIC Library Files

An ASIC Library Data Format 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.

Output Files

The synthesis tool generates reports about the synthesis run and files that you can use for simulation or placement and routing. The following table describes the output files, categorizing them as either synthesis result and report files, or output files generated as input for other tools.

Extension	File	Description
.areasrr	Hierarchical Area Report	Reports area-specific information such as sequential and combinational ATOMS, RAMs, DSPs, and Black Boxes on each module in the design. See Hierarchical Area Report, on page 267 .
_cck.rpt	Constraint Checker Report	Checks the syntax and applicability of the timing constraints in the fdc file for your project and generates a report (<i>projectName_cck.rpt</i>). See Constraint Checking Report, on page 268 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.

Extension	File	Description
.linkerlog	Mixed language ports/generics differences	Provides details of why the VHDL and/or Verilog components in the source files were not properly linked. This file is located in the synwork directory for the implementation. The same information is also reported in the log file.
.pfl	Message Filter criteria	Output file created after filtering messages in the Messages window. See Updating the projectName.pfl file, on page 206 in the <i>User Guide</i> .
Results file: • .edf	Vendor-specific results file	Results file that contains the synthesized netlist, written out in a format appropriate to the technology and the place-and-route tool you are using. Generally, the format is EDIF. Specify this file on the Implementation Results panel of the Implementation Options dialog box (Implementation Results Panel, on page 210).
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 34 .

Extension	File	Description
.sap	Synplify Annotated Properties	This file is generated after the Annotated Properties for Analyst option is selected in the Device panel of the Implementation Options dialog box. After the compile stage, the tool annotates the design with properties like clock pins. You can find objects based on these annotated properties using Tcl Find. For more information, see find, on page 92 Using the Tcl Find Command to Define Collections, on page 142 in the <i>User Guide</i> .
.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 196 for details on archiving, unarchiving and copying projects.
_sckk.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 254 , for more information.

Extension	File	Description
.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.
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 178 .
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 178 .
.ta	Customized Timing Report	Contains the custom timing information that you specify through Analysis->Timing Analyst. See Analysis Menu, on page 281 , 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 281 for more information.

Extension	File	Description
.tap	Timing Annotated Properties	This file is generated after the Annotated Properties for Analyst option is selected in the Device panel of the Implementation Options dialog box. After the compile stage, the tool annotates the design with timing properties and the information can be analyzed in the RTL view and Design Planner. You can also find objects based on these annotated properties using Tcl Find. For more information, see Using the Tcl Find Command to Define Collections, on page 142 in the <i>User Guide</i> .
.tlg	Log file	This log file contains a list of all the modules compiled in the design.
<i>vendor constraint file</i>	Constraints file for forward annotation	Contains synthesis constraints to be forward-annotated to the place-and-route tool. The constraint file type varies with the vendor and the technology. Refer to the vendor chapters for specific information about the constraints you can forward-annotate. Check the Implementation Results dialog (Implementation Options) for supported files. See Implementation Results Panel, on page 210 .

Extension	File	Description
.vm .vhm	Mapped Verilog or VHDL netlist	<p>Optional post-synthesis netlist file in Verilog (.vm) or VHDL (.vhm) format. This is a structural netlist of the synthesized design, and differs from the original RTL used as input for synthesis. Specify these files on the Implementation Results dialog box (Implementation Options). See Implementation Results Panel, on page 210.</p> <p>Typically, you use this netlist for gate-level simulation, to verify your synthesis results. Some designers prefer to simulate before and after synthesis, and also after place-and-route. This approach helps them to isolate the stage of the design process where a problem occurred.</p> <p>The Verilog and VHDL output files are for functional simulation only. When you input stimulus into a simulator for functional simulation, use a cycle time for the stimulus of 1000 time ticks.</p>

Log File

The log file report, located in the implementation directory, is written out in two file formats: text (*projectName.srr*), and HTML with an interactive table of contents (*projectName.htm* and *projectName_srr.htm*) where *projectName* is the name of your project. Select View Log File in HTML in the Options->Project View Options dialog box to enable viewing the log file in HTML. Select the View Log button in the Project view ([Buttons and Options, on page 92](#)) to see the log file report.

The log file is written each time you compile or synthesize (compile and map) the design. When you compile a design without mapping it, the log file contains only compiler information. As a precaution, a backup copy of the log file (srr) is written to the backup sub-directory in the Implementation Results directory. Only one backup log file is updated for subsequent synthesis runs.

The log file contains detailed reports on the compiler, mapper, timing, and resource usage information for your design. Errors, notes, warnings, and messages appear in both the log file and on the Messages tab in the Tcl window.

For further details about different sections of the log file, see the following:

For information about ...	See ...
Compiled files, messages (warnings, errors, and notes), user options set for synthesis, state machine extraction information, including a list of reachable states.	Compiler Report, on page 256
Buffers added to clocks in certain supported technologies.	Clock Buffering Report, on page 256
Buffers added to nets.	Net Buffering Report, on page 257
Compile point remapping	Compile Point Information, on page 257
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 259
Resources used by synthesis mapping	Resource Usage Report, on page 258
Design changes made as a result of retiming	Retiming Report, on page 258

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.

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 433](#) of the *User Guide*.

Timing Section

A default timing report is written to the log file (*projectName.srr*) in the “START OF TIMING REPORT” section. See [Timing Reports, on page 259](#), for details.

For certain device technologies, you can use the Timing Analyst to generate additional timing reports for point-to-point analysis (see [Analysis Menu, on page 281](#)). 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 195](#) 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:

<code>.srr</code> or <code>.htm</code>	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.
<code>.ta</code>	Timing analysis file that contains timing information based on the parameters you specify in the stand-alone Timing Analyst (Analysis->Timing Analyst).
<code>designName_async_clk</code> <code>.rpt.scv</code>	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 266 for details on this report.

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

- [Timing Report Header, on page 260](#)
- [Performance Summary, on page 260](#)
- [Clock Relationships, on page 262](#)
- [Interface Information, on page 263](#)
- [Detailed Clock Report, on page 264](#)
- [Asynchronous Clock Report, on page 266](#)

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.

```
00055 ##### START TIMING REPORT #####
00056 ##### START TIMING REPORT #####
00057 # Timing Report written on Fri Sep 06 13:38:15 2002
00058 #
00059
00060
00061 Top view:                mod2
00062 Paths requested:         5
00063 Constraint File(s):
00064 [N] This timing report estimates place and route data. Please look :
00065 [N] Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF
00066
```

You can control the size of the timing report by choosing Project -> Implementation Options, clicking the Timing Report tab of the panel, and specifying the number of start/end points and the number of critical paths to report. See [Timing Report Panel, on page 212](#), 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 264](#)). The Performance Summary lists the following information for each clock in the design:

Performance Summary Column	Description
Starting Clock	Clock at the start point of the path. If the clock name is system, the clock is a collection of clocks with an undefined clock event. Rising and falling edge clocks are reported as one clock domain.
Requested/Estimated Frequency	Target frequency goal /estimated value after synthesis. See Cross-Clock Path Timing Analysis, on page 262 for information on how cross-clock path slack is reported.
Requested/Estimated Period	Target clock period/estimated value after synthesis.

Performance Summary Column	Description
Slack	Difference between estimated and requested period. See Cross-Clock Path Timing Analysis, on page 262 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 261 .
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.

- 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 262](#) 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 155](#). For tips on using clock groups, see [Defining Other Clock Requirements, on page 171](#) in the *User Guide*.

Clock Relationships

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

Cross-Clock Path Timing Analysis

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

Cross-Clock Path Frequency

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

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

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

It applies relevant multicycle constraints to the setup calculation.

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

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

Slack for Cross-Clock Paths

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

Interface Information

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

Port parameter	Description
Port Name	Port name.
Starting Reference Clock	The reference clock.
User Constraint	The input/output delay. If a port has multiple delay records, the report contains the values for the record with the worst slack. The reference clock corresponds to the worst slack delay record.
Arrival Time	Input ports: <code>define_input_delay</code> , 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 + <code>define_reg_input_delay</code> value). Output ports: clock period – <code>define_output_delay</code> . Default value of <code>define_output_delay</code> 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
<i>N</i> most critical start points	Start points can be input ports or registers. If the start point is a register, you see the starting pin in the report. To change the number of start points reported, choose Project -> Implementation Options, and set the number on the Timing Report panel.

Clock Information	Description
<i>N</i> 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.
<i>N</i> 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 260](#)) is just the clock period.

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

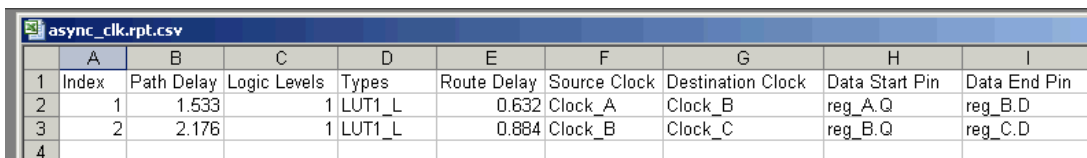
Critical path information	Description
Instance/Net Name	Technology view names for the instances and nets in the critical path
Type	Type of cell
Pin Name	Name of the pin
Pin Dir	Pin direction
Delay	The delay value.
Arrival Time	Clock delay at the source + the propagation delay through the path
Fan Out	Number of fanouts for the point in the path

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 (τ_a) 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 τ_a
Source Clock	Start clock.
Destination Clock	End clock.
Data Start Pin	Sequential device output pin at start of path.
Data End Pin	Setup check pin at destination.



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

Hierarchical Area Report

An area report is created during synthesis which contains the percentage utilization for elements in the design, as well as, total sequential utilization for elements of specific modules. For instance, elements can include sequential, combinational, or memory elements. They can also include the following types of technology-specific elements for ROMs, I/O pads, or DSPs.

This report generates technology-specific area information that is reflected in the output depending upon the specified device. The report is written to the *projectName*.areasrr file. You can view the file with the log viewer or any text editor.

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

This section describes the following topics:

- [Reporting Details, on page 268](#)
- [Inapplicable Constraints, on page 269](#)
- [Applicable Constraints With Warnings, on page 270](#)
- [Sample Constraint Check Report, on page 271](#)

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 269 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 270 for more information.
Constraints with matching wildcard expressions	Lists constraints or collections using wildcard expressions up to the first 1000, respectively.

Inapplicable Constraints

Refer to the following table for constraints that were not applied because objects do not exist or the object type check was not valid:

For these constraints ...	Objects must be ...
Attributes	Valid definitions
create_clock	<ul style="list-style-type: none"> • Ports • Nets • Pins • Registers • Instantiated buffers
create_generated_clock	Clocks
define_compile_point	<ul style="list-style-type: none"> • Region • View
define_current_design	v: view

For these constraints ...	Objects must be ...
set_false_path set_multicycle_path set_max_delay	For -to or -from objects: <ul style="list-style-type: none"> • i:sequential instances • p:ports • i:black boxes For -through objects <ul style="list-style-type: none"> • n:nets • t:hierarchical ports • t:pins
set_multicycle_path	Specified as a positive integer
set_input_delay	<ul style="list-style-type: none"> • Input ports • bidir ports
set_output_delay	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • i:sequential instances • p:ports • i:black boxes For -through objects: <ul style="list-style-type: none"> • 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 2013
# Copyright (C) 1994-2013, Synopsys, Inc.

# Written on Fri Jun 7 09:42:22 2013
##### 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

CHAPTER 8

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 278](#)
- [Block RAM Examples, on page 279](#)
- [Initial Values for RAMs, on page 292](#)
- [RAM Instantiation with SYNCORE, on page 297](#)
- [ROM Inference, on page 298](#)

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	Microsemi
Single Port	x
Dual Port	x
True Dual Port	x

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 279](#)
- [Single-Port Block RAM Examples, on page 283](#)
- [Dual-Port Block RAM Examples, on page 286](#)
- [True Dual-Port RAM Examples, on page 288](#)

For details about inferring block RAM, see [Automatic RAM Inference, on page 367](#) in the *User Guide*.

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 279](#)
- [READ_FIRST Mode Example, on page 281](#)
- [NO_CHANGE Mode Example, on page 282](#)

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;
```

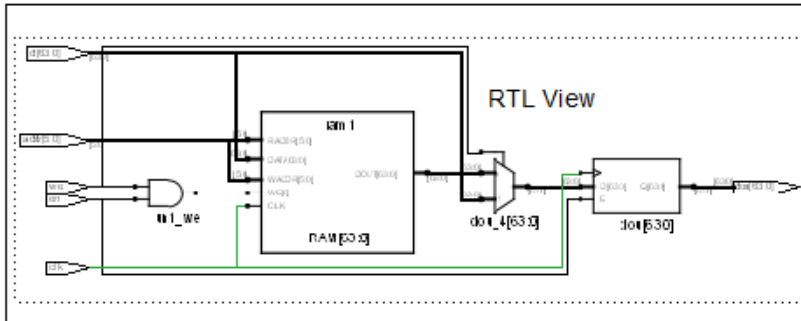
```

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. The Technology view shows that the RAM is mapped to a block RAM.



READ_FIRST Mode Example

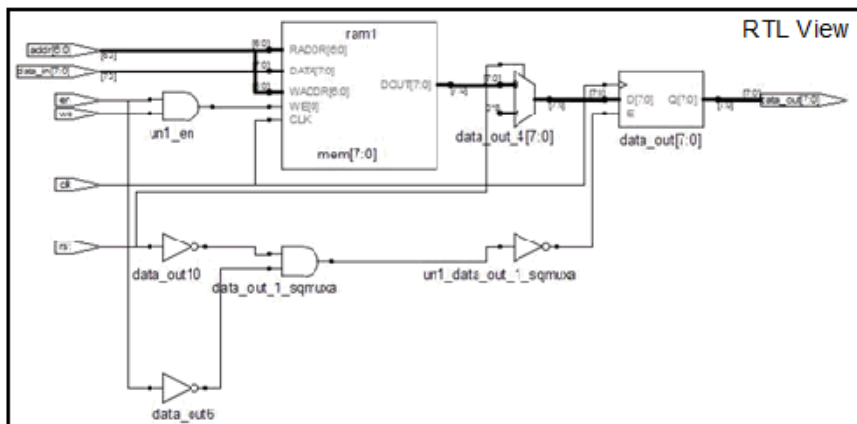
The following piece of code is an example of READ_FIRST mode with both enable and reset, with reset taking precedence:

```
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. The Technology view shows 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;

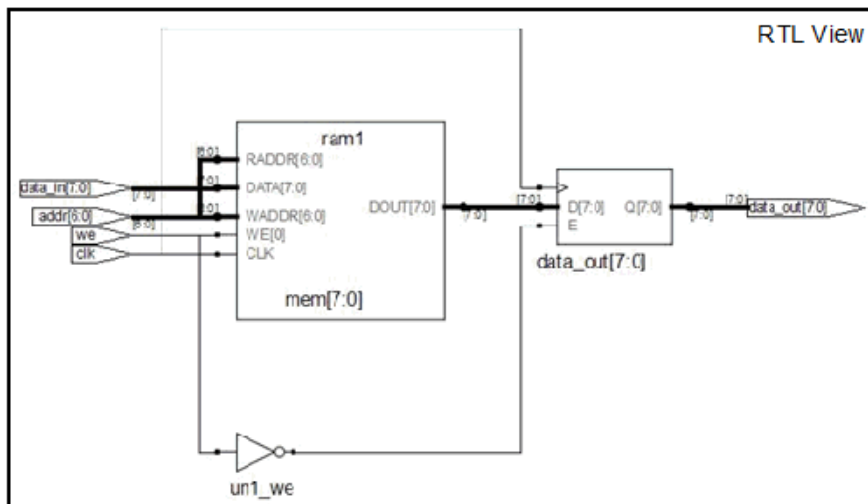
always@(posedge clk)
if(we == 1)
    data_out <= data_in;
else
    data_out <= mem[addr];

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

endmodule

```

The next figure shows the RTL view of a NO_CHANGE RAM. The Technology view shows 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 283](#)
- [Single-Port RAM with RAM Output Registered Examples, on page 284](#)
- [Dual-Port Block RAM Examples, on page 286](#)

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
  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.

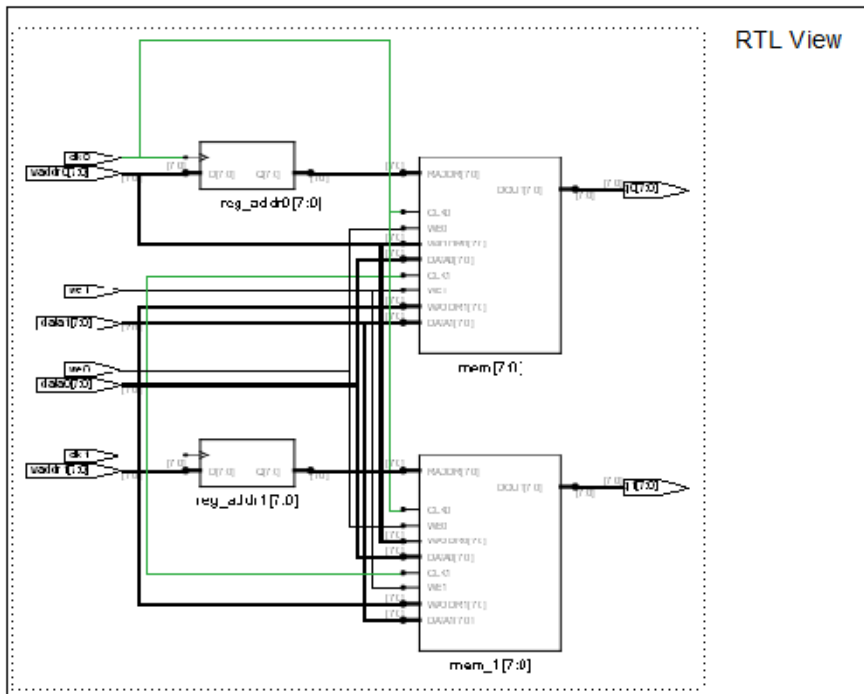
Verilog Example: True Dual-Port RAM

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]]);
```

<code>\$readmemb</code>	Use this with a binary data file.
<code>\$readmemh</code>	Use this with a hexadecimal data file.
<i>fileName</i>	Name of the data file that contains initial values. See Initialization Data File, on page 294 for format examples.
<i>memoryName</i>	The name of the memory.
<i>startAddress</i>	Optional starting address for RAM initialization; if omitted, defaults to first available memory location.
<i>stopAddress</i>	Optional stopping address for RAM initialization; <i>startAddress</i> must be specified

Also, see the following topics:

- [Example 1: RAM Initialization, on page 292](#)
- [Example 2: Cross-Module Referencing for RAM Initialization, on page 293](#)
- [Initialization Data File, on page 294](#)
- [Forward Annotation of Initial Values, on page 297](#)

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 294](#) for details of the binary and hexadecimal file formats.

```

module ram_inference (data, clk, addr, we, data_out);
input [27:0] data;
input clk, we;
input [10:0] addr;
output [27:0] data_out;
reg [27:0] mem [0:2000] /* synthesis syn_ramstyle = "no_rw_check" */;
reg [10:0] addr_reg;

initial
begin
    $readmemb ("mem.ini", mem, 2, 1900) /* Initialize RAM with contents */
    /* from locations 2 thru 1900*/;
end

always @(posedge clk)
begin
    addr_reg <= addr;
end

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

assign data_out = mem[addr_reg];
endmodule

```

Example 2: Cross-Module Referencing for RAM Initialization

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

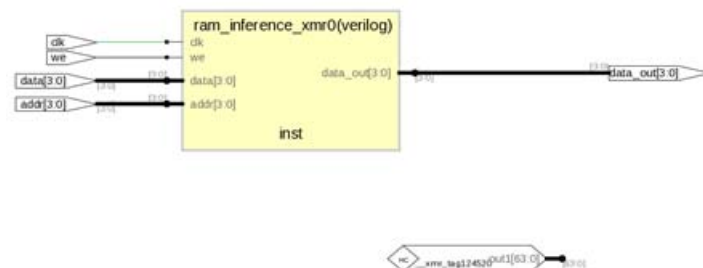
Example2A: XMR for RAM Initialization (Top-Level Module)

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

Example2B: XMR for RAM Initialization (RAM)

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.



Limitations

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 296](#)), 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

Binary File Format

```
11111111111111111111111100110111 /* data for address 0 */
11111111111111111111111101100111 /* data for address 1 */
1111111111111111111111111000010
111111111111111111111111100100001
11111111111111111111111101110000
111111111111111111111111011100110
... /* continues until Address 1999 */
```

```

FFFFF37      /* data for address 0 */
FFFFF63      /* data for address 1 */
FFFFFC2
FFFFF21
.../* continues until Address 1999 */

```

```
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 456 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 9

IP and Encryption Tools

This chapter describes the SYNCORE IP functionality that is bundled with the synthesis tools, and supported IP encryption standards.

- [SYNCORE FIFO Compiler, on page 304](#)
- [SYNCORE RAM Compiler, on page 321](#)
- [SYNCORE Byte-Enable RAM Compiler, on page 331](#)
- [SYNCORE ROM Compiler, on page 336](#)
- [SYNCORE Adder/Subtractor Compiler, on page 342](#)
- [SYNCORE Counter Compiler, on page 354](#)
- [Encryption Scripts, on page 359](#)

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 304](#)
- [FIFO Read and Write Operations, on page 305](#)
- [FIFO Ports, on page 307](#)
- [FIFO Parameters, on page 309](#)
- [FIFO Status Flags, on page 311](#)
- [FIFO Programmable Flags, on page 314](#)

For further information, refer to the following:

- [Specifying FIFOs with SYNCore, on page 450](#) of the *User Guide*, for information about using the wizard to generate FIFOs
- [Launch SYNCore Command, on page 242](#) and [SYNCore FIFO Wizard, on page 244](#) 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 305](#) 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 306](#) 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 311](#) and [FIFO Programmable Flags, on page 314](#).

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

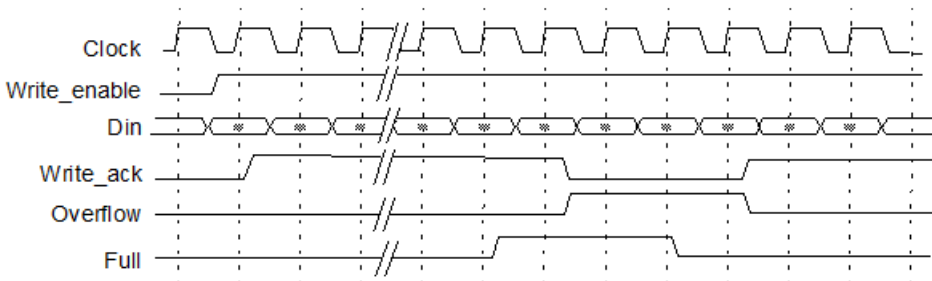
This section describes FIFO behavior with read and write operations.

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.

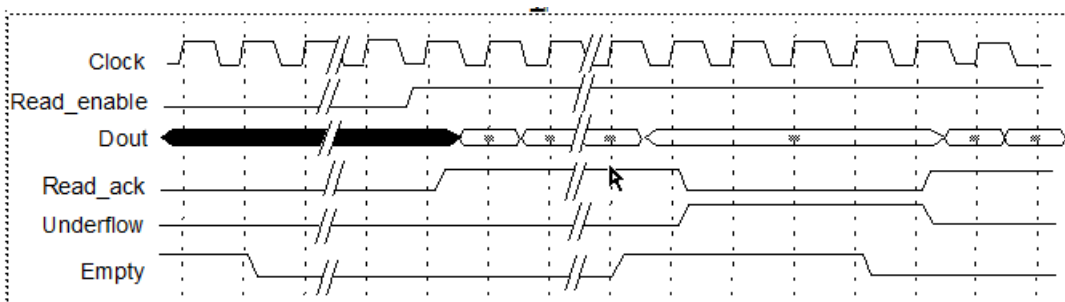


FIFO Read Operation

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

Port Name	Description
Empty	FIFO empty output (active high). Asserted when the FIFO is empty and no additional reads can be performed. Can be active high or active low.
Full	FIFO full output (active high). Asserted when the FIFO is full and no additional writes can be performed. Can be active high or active low.
Overflow	FIFO overflow output flag (active high). Asserted when the FIFO is full and the previous write was rejected. Can be active high or active low.
Prog_empty	Programmable empty output flag (active high). Asserted when the words in the FIFO exceed or equal the programmable empty assert threshold. De-asserted when the number of words is more than the programmable full negate threshold. Can be active high or active low.
Prog_empty_thresh	Programmable FIFO empty threshold input. User-programmable threshold value for the assertion of the Prog_empty flag. Set during reset.
Prog_empty_thresh_assert	Programmable FIFO empty threshold assert input. User-programmable threshold value for the assertion of the Prog_empty flag. Set during reset.
Prog_empty_thresh_negate	Programmable FIFO empty threshold negate input. User-programmable threshold value for the de-assertion of the Prog_full flag. Set during reset.
Prog_full	Programmable full output flag (active high). Asserted when the words in the FIFO exceed or equal the programmable full assert threshold. De-asserted when the number of words is less than the programmable full negate threshold. Can be active high or active low.
Prog_full_thresh	Programmable FIFO full threshold input. User-programmable threshold value for the assertion of the Prog_full flag. Set during reset.
Prog_full_thresh_assert	Programmable FIFO full threshold assert input. User-programmable threshold value for the assertion of the Prog_full flag. Set during reset.
Prog_full_thresh_negate	Programmable FIFO full threshold negate input. User-programmable threshold value for the de-assertion of the Prog_full flag. Set during reset.

Port Name	Description
Read_ack	Read acknowledge output (active high). Asserted when valid data is read from the FIFO. Can be active high or active low.
Read_enable	Read enable output (active high). If the FIFO is not empty, data is read from the FIFO on the next rising edge of the read clock.
Underflow	FIFO underflow output flag (active high). Asserted when the FIFO is empty and the previous read was rejected.
Write_ack	Write Acknowledge output (active high). Asserted when there is a valid write into the FIFO. Can be active high or active low.
Write_enable	Write enable input (active high). If the FIFO is not full, data is written into the FIFO on the next rising edge.

FIFO Parameters

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

Parameter	Description
PEMPTY_FLAG_SENSE	FIFO programmable empty flag sense 0 Active Low 1 Active High
PFULL_FLAG_SENSE	FIFO programmable full flag sense 0 Active Low 1 Active High
PGM_EMPTY_ATHRESH	Programmable empty assert threshold for PGM_EMPTY_TYPE=2
PGM_EMPTY_NTHRESH	Programmable empty negate threshold for PGM_EMPTY_TYPE=2
PGM_EMPTY_THRESH	Programmable empty threshold for PGM_EMPTY_TYPE=1
PGM_EMPTY_TYPE	Programmable empty type. See Programmable Empty, on page 318 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 315 for details. 1 Programmable full with single threshold constant 2 Programmable full with multiple threshold constant 3 Programmable full with single threshold input 4 Programmable full with multiple threshold input
RACK_FLAG_SENSE	FIFO read acknowledge flag sense 0 Active Low 1 Active High

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

FIFO Status Flags

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

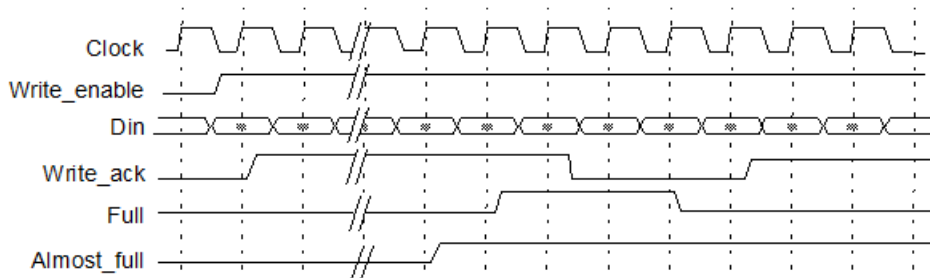
- [Full/Almost Full Flags, on page 311](#)
- [Empty/Almost Empty Flags, on page 312](#)
- [Handshaking Flags, on page 313](#)
- Programmable full and empty flags, which are described in [Programmable Full, on page 315](#) and [Programmable Empty, on page 318](#).

Full/Almost Full Flags

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

Full	Indicates that the FIFO memory queue is full and no more writes can be performed until data is read. Full is synchronous with the clock (Clock). If a write is initiated when Full is asserted, the write does not succeed and the overflow flag is asserted.
Almost_full	The almost full flag (Almost_full) indicates that there is one location left and the FIFO will be full after one more write operation. Almost full is synchronous to Clock. This flag is guaranteed to be asserted when the FIFO has one remaining location for a write operation.

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

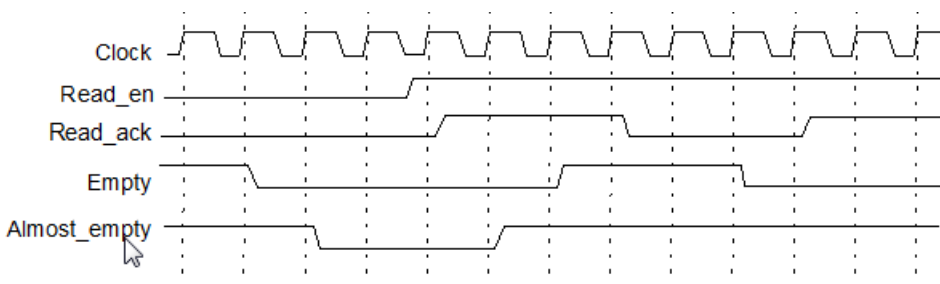


Empty/Almost Empty Flags

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

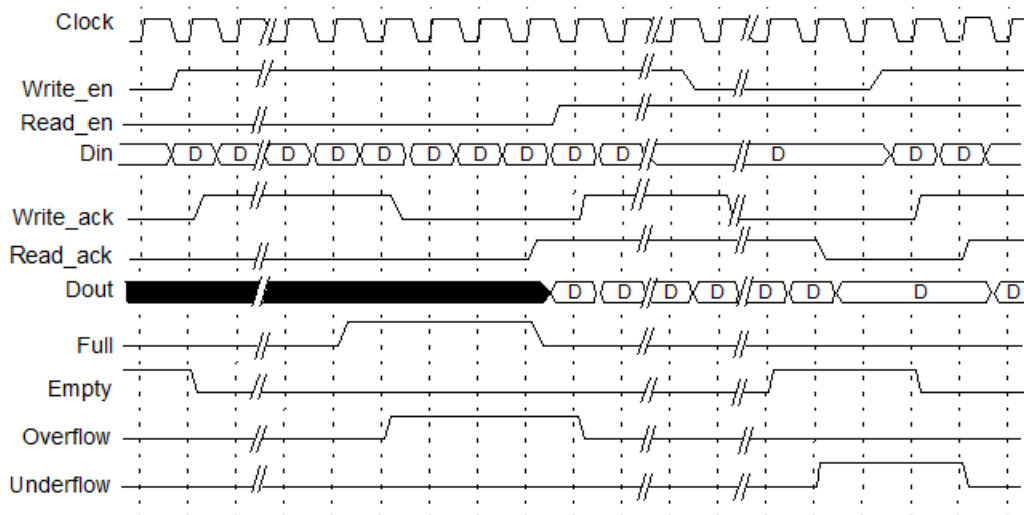
Empty	Indicates that the memory queue for the FIFO is empty and no more reads can be performed until data is written. The output is active high and is synchronous to the clock. If a read is initiated when the empty flag is true, the underflow flag is asserted.
Almost_empty	Indicates that the FIFO will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

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



Handshaking Flags

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



Read_ack	<p>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.</p> <p>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.</p>
Write_ack	<p>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.</p> <p>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.</p>
Overflow	<p>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.</p>
Underflow	<p>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.</p>

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 315 for more information.
Prog_empty	Indicates that the FIFO has reached a user-defined empty threshold. See Programmable Empty, on page 318 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

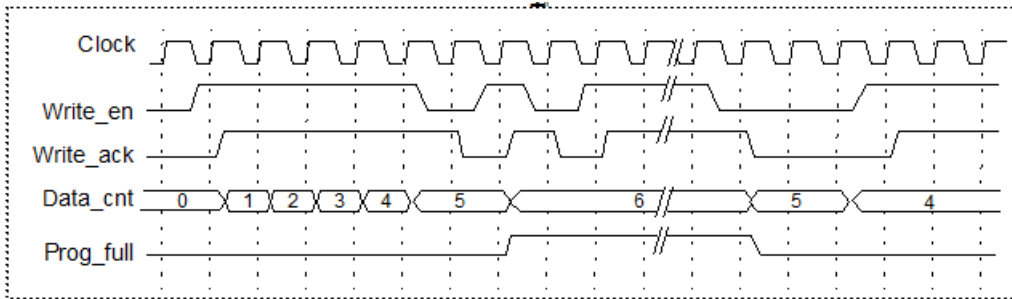
Prog_full has four threshold types:

- [Programmable Full with Single Threshold Constant, on page 315](#)
- [Programmable Full with Multiple Threshold Constants, on page 316](#)
- [Programmable Full with Single Threshold Input, on page 316](#)
- [Programmable Full with Multiple Threshold Inputs, on page 317](#)

Programmable Full with Single Threshold Constant

PGM_FULL_TYPE = 1

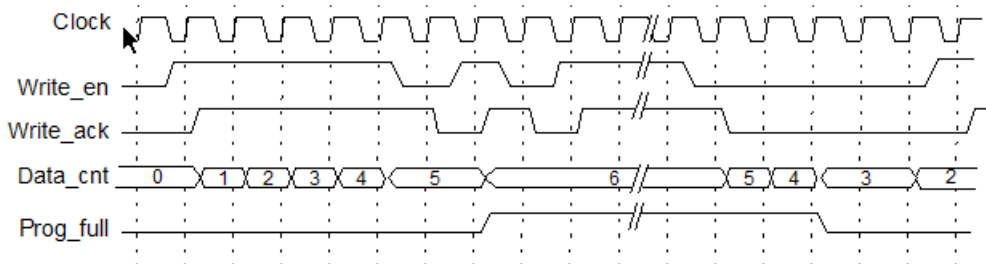
This option lets you set a single constant value for the threshold. It requires significantly fewer resources when the FIFO is generated. This figure illustrates the behavior of Prog_full when configured as a single threshold constant with a value of 6.



Programmable Full with Multiple Threshold Constants

`PGM_FULL_TYPE = 2`

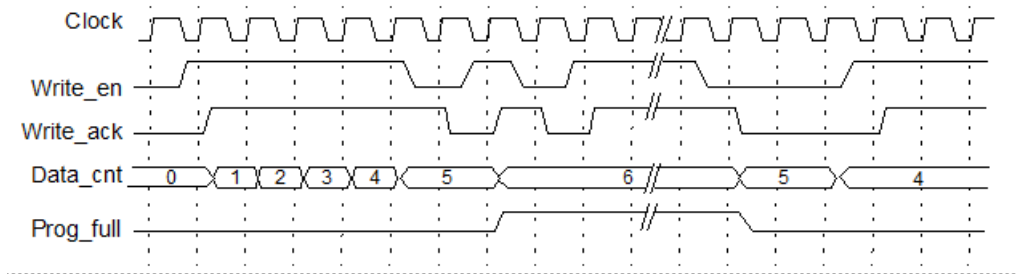
The programmable full flag is asserted when the number of words in the FIFO is greater than or equal to the full threshold assert value. If the number of FIFO words drops to less than the full threshold negate value, the programmable full flag is de-asserted. Note that the negate value must be set to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

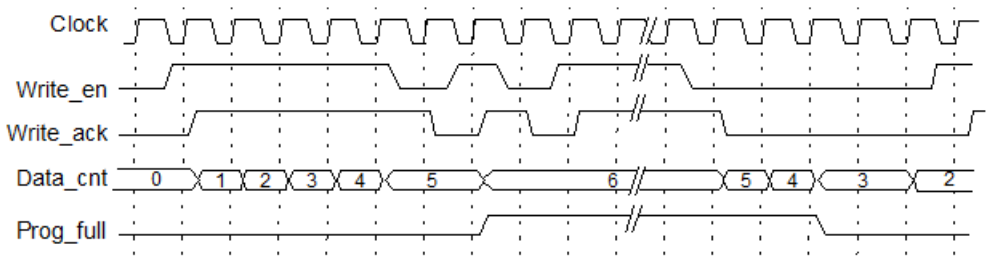


Programmable Full with Multiple Threshold Inputs

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

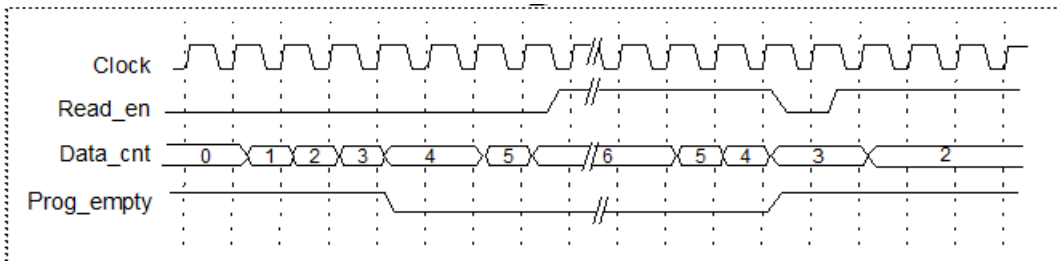
There are four threshold types you can specify:

- [Programmable Empty with Single Threshold Constant, on page 318](#)
- [Programmable Empty with Multiple Threshold Constants, on page 319](#)
- [Programmable Empty with Single Threshold Input, on page 319](#)
- [Programmable Empty with Multiple Threshold Inputs, on page 320](#)

Programmable Empty with Single Threshold Constant

PGM_EMPTY_TYPE = 1

This option lets you specify an empty threshold value with a single constant. This approach requires significantly fewer resources when the FIFO is generated. The following figure illustrates the behavior of Prog_empty configured as a single threshold constant with a value of 3.

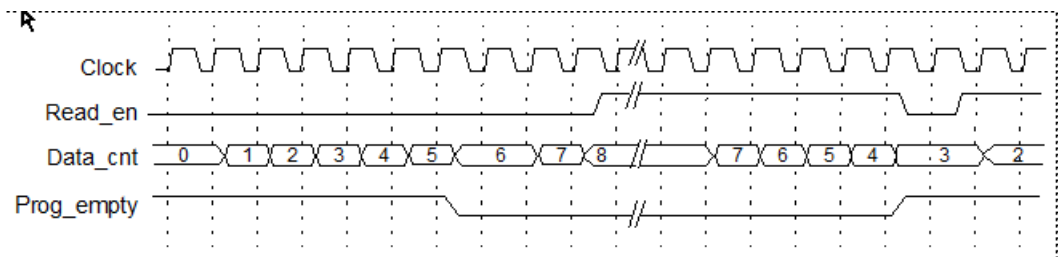


Programmable Empty with Multiple Threshold Constants

PGM_EMPTY_TYPE = 2

This option lets you specify constants for the empty threshold assert value and empty threshold negate value. The programmable empty flag asserts and deasserts in the range set by the assert and negate values. The assert value must be set to a value less than the negate value. When the number of words in the FIFO is less than or equal to the empty threshold assert value, the Prog_empty flag is asserted. When the number of words in FIFO is greater than the empty threshold negate value, Prog_empty is deasserted.

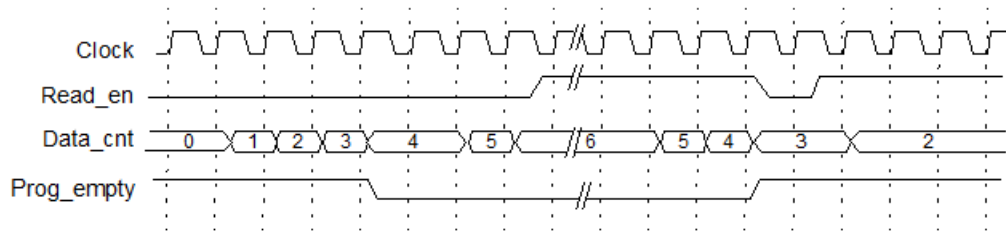
The following figure illustrates the behavior of Prog_empty when configured as multiple threshold constants with an assert value of 3 and a negate value of 5.



Programmable Empty with Single Threshold Input

PGM_EMPTY_TYPE = 3

This option lets you specify the threshold value dynamically during the reset state with the Prog_empty_thresh input port, instead of with a constant. The Prog_empty flag asserts when the number of FIFO words is equal to or less than the Prog_empty_thresh value and deasserts when the number of FIFO words is more than the Prog_empty_thresh value. The following figure illustrates the behavior of Prog_empty when configured as a single threshold input with a value of 3.

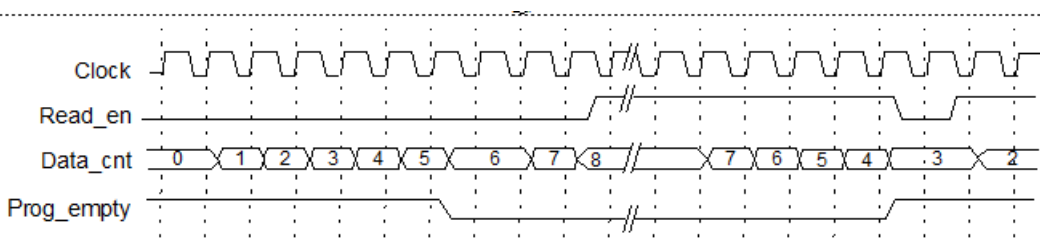


Programmable Empty with Multiple Threshold Inputs

PGM_EMPTY_TYPE = 4

This option lets you specify the assert and negate threshold values dynamically during the reset stage using the `Prog_empty_thresh_assert` and `Prog_empty_thresh_negate` input ports instead of constants. The programmable empty flag asserts and deasserts according to the range set by the assert and negate values. The assert value must be set to a value less than the negate value.

When the number of FIFO words is less than or equal to the empty threshold assert value, `Prog_empty` is asserted. If the number of FIFO words is greater than the empty threshold negate value, the flag is deasserted. The following figure illustrates the behavior of `Prog_empty` configured as multiple threshold inputs, with an assert value of 3 and a negate value of 5.



SYNCore RAM Compiler

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

- [Single-Port Memories, on page 321](#)
- [Dual-Port Memories, on page 323](#)
- [Read/Write Timing Sequences, on page 328](#)

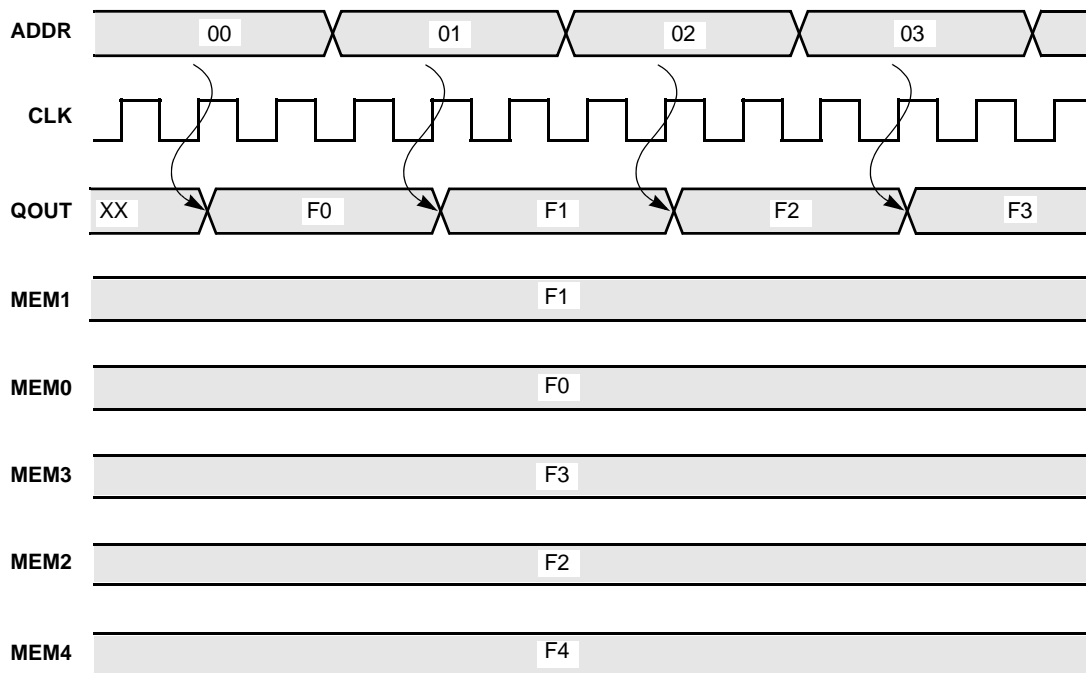
For further information, refer to the following:

- [Specifying RAMs with SYNCore, on page 456](#) of the *User Guide*, for information about using the wizard to generate FIFOs
- [Launch SYNCore Command, on page 242](#) and [SYNCore FIFO Wizard, on page 244](#) for descriptions of the interface

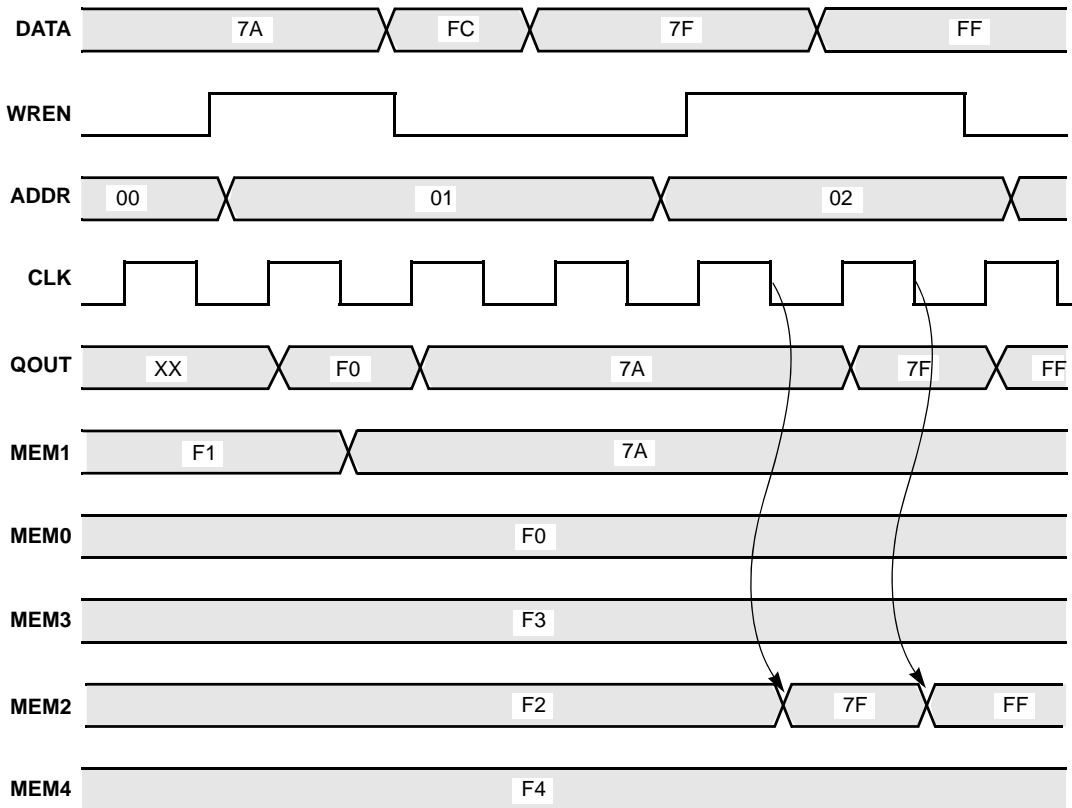
Single-Port Memories

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 456](#) in the *User Guide* for a procedure.

Single-Port Read



Single-Port Write



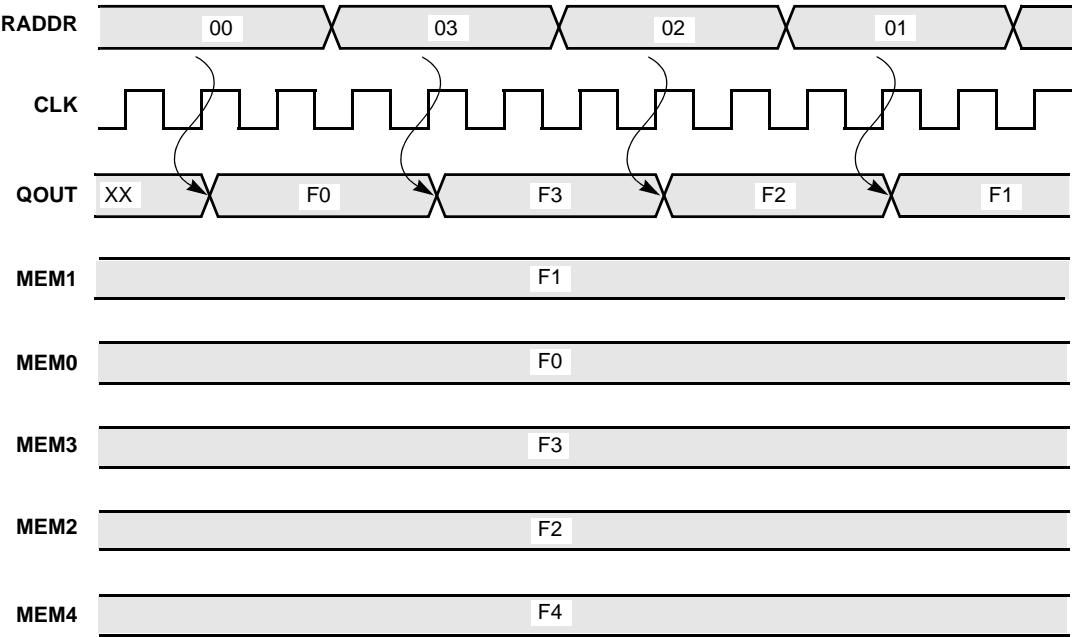
Dual-Port Memories

SYNCore dual-port memory includes the following common configurations:

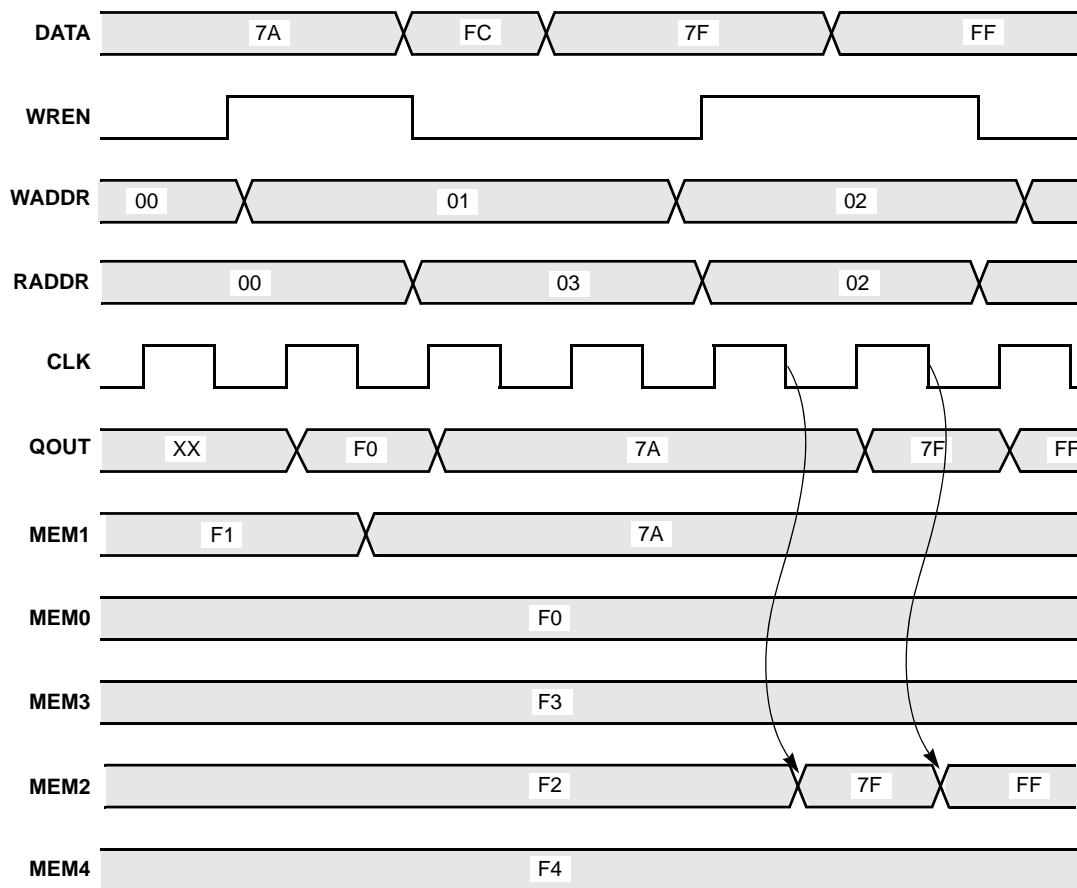
- One read access and one write access
- Two read accesses and one write access
- Two read accesses and two write accesses

The following diagrams show the read-write timing for dual-port memories. See [Specifying RAMs with SYNCore, on page 456](#) in the *User Guide* for a procedure to specify a dual-port RAM with SYNCore.

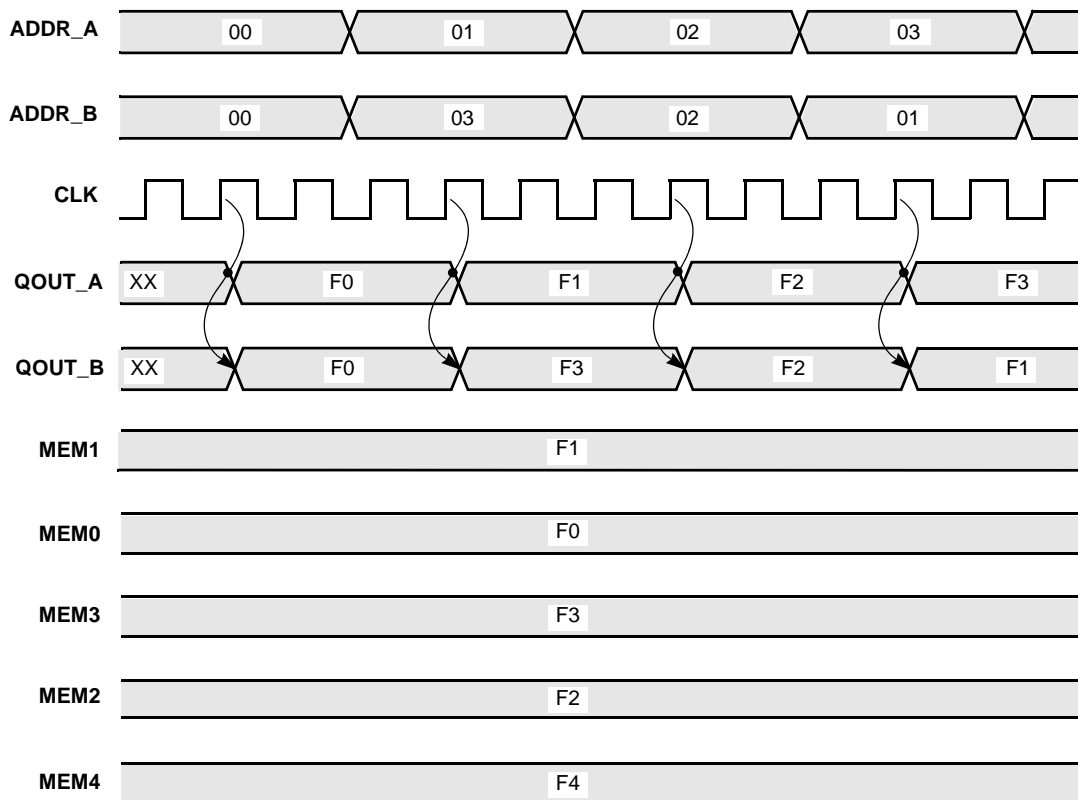
Dual-Port Single Read



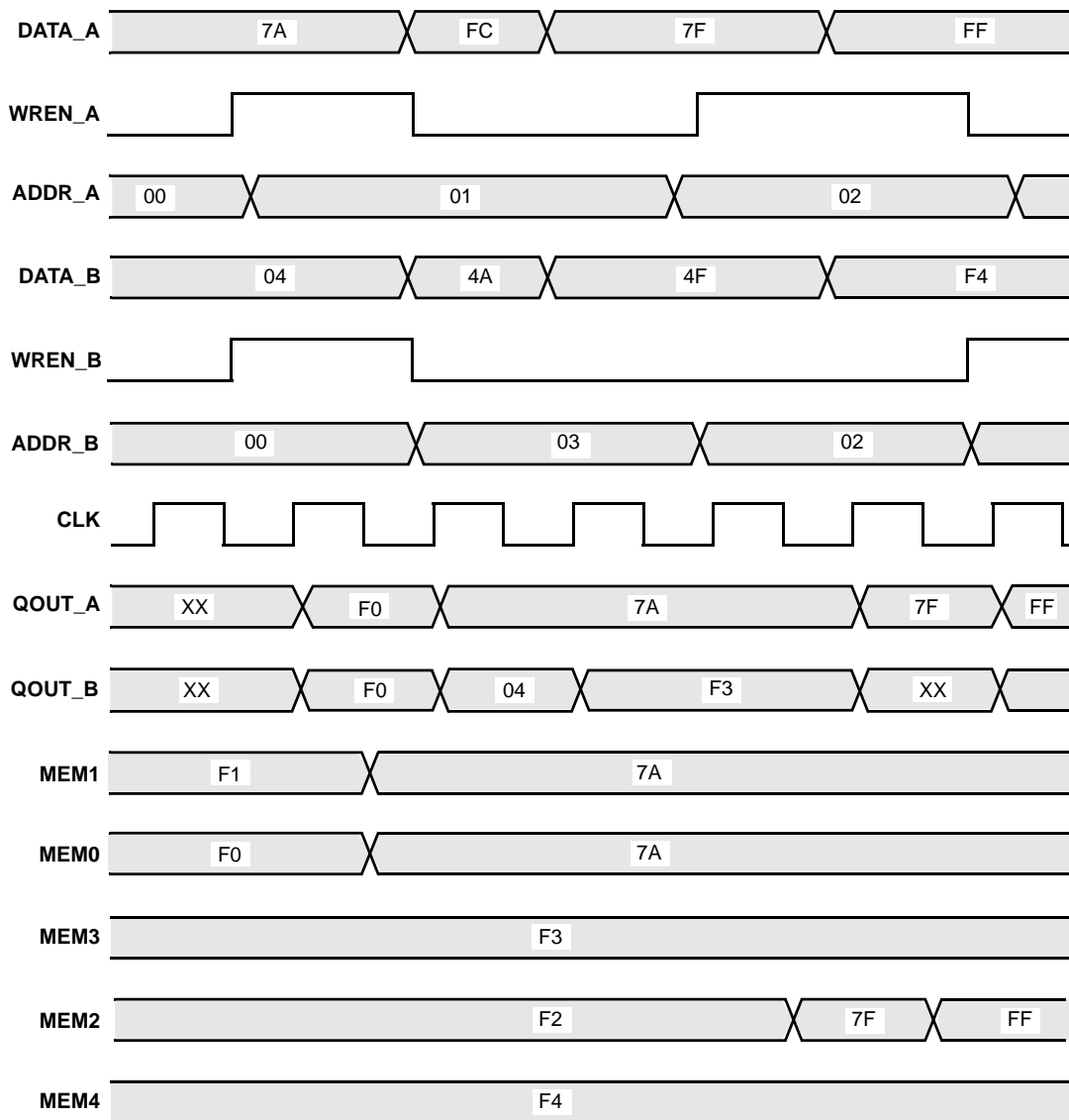
Dual-Port Single Write



Dual-Port Read



Dual-Port Write

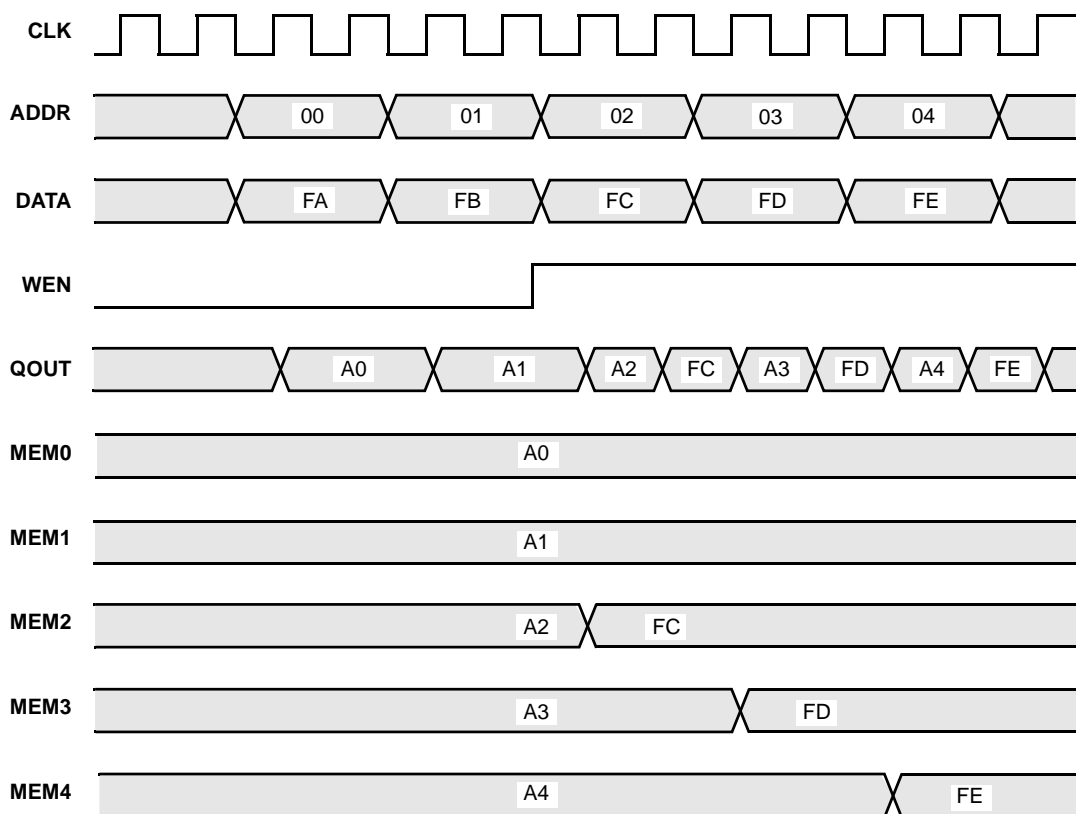


Read/Write Timing Sequences

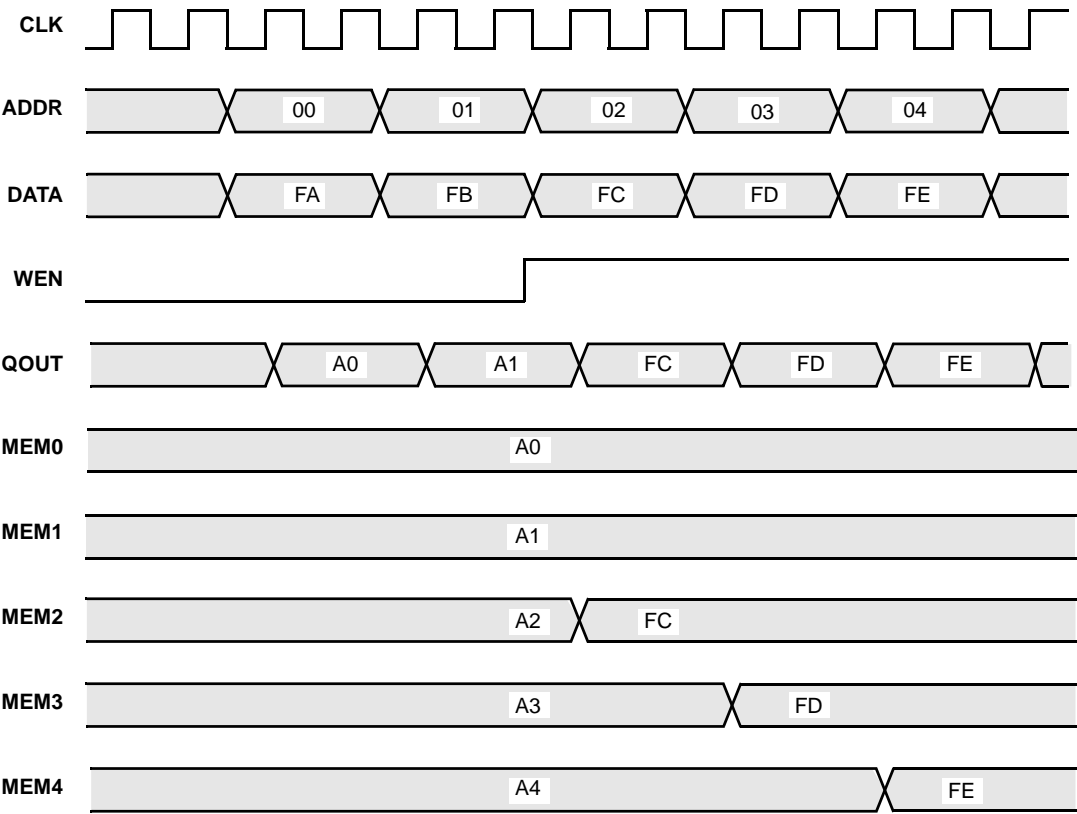
The waveforms in this section describe the behavior of the RAM when both read and write are enabled and the address is the same operation. The waveforms show the behavior when each of the read-write sequences is enabled. The waveforms are merged with the simple waveforms shown in the previous sections.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

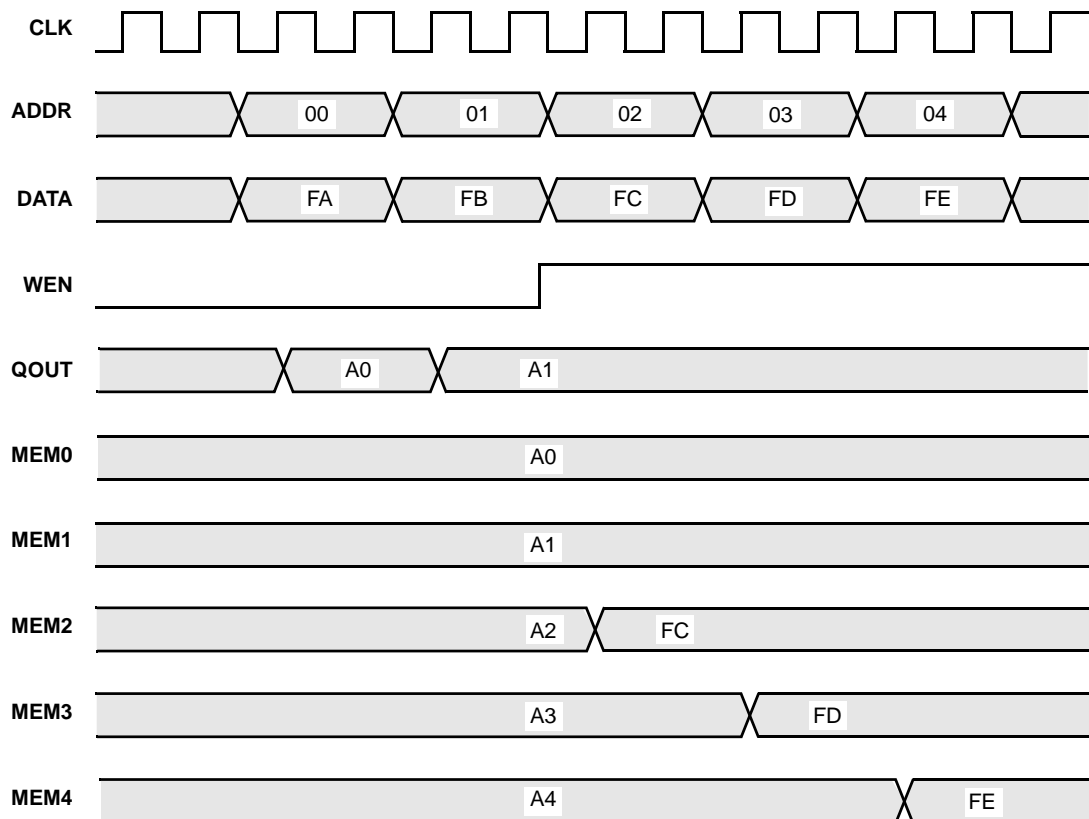
- [Read Before Write, on page 328](#)
- [Write Before Read, on page 329](#)
- [No Read on Write, on page 330](#)

Read Before Write



Write Before Read



No Read on Write

SYNCore Byte-Enable RAM Compiler

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

This section describes the following:

- [Functional Overview, on page 331](#)
- [Read Operation, on page 332](#)
- [Write Operation, on page 333](#)
- [Parameter List, on page 335](#)

For further information, refer to the following:

- [Specifying Byte-Enable RAMs with SYNCore, on page 464](#) 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 258](#) 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.

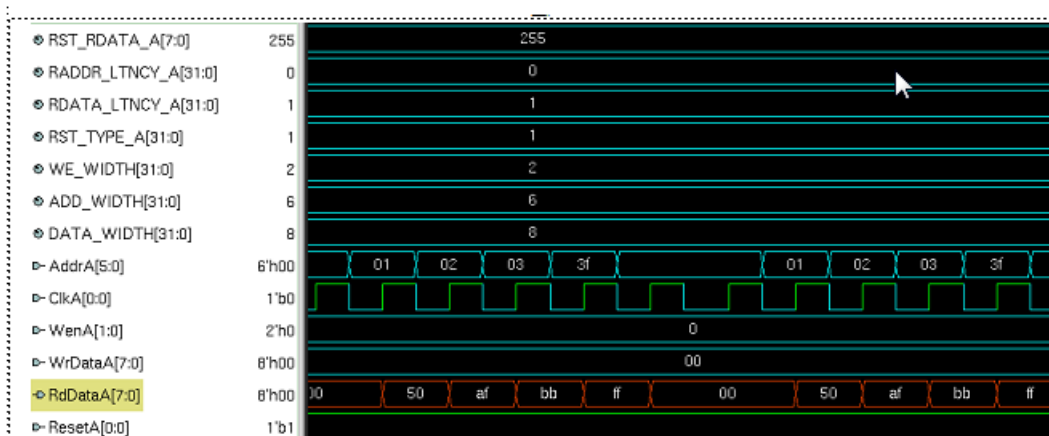
Read/Write Timing Sequences

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

Read Operation

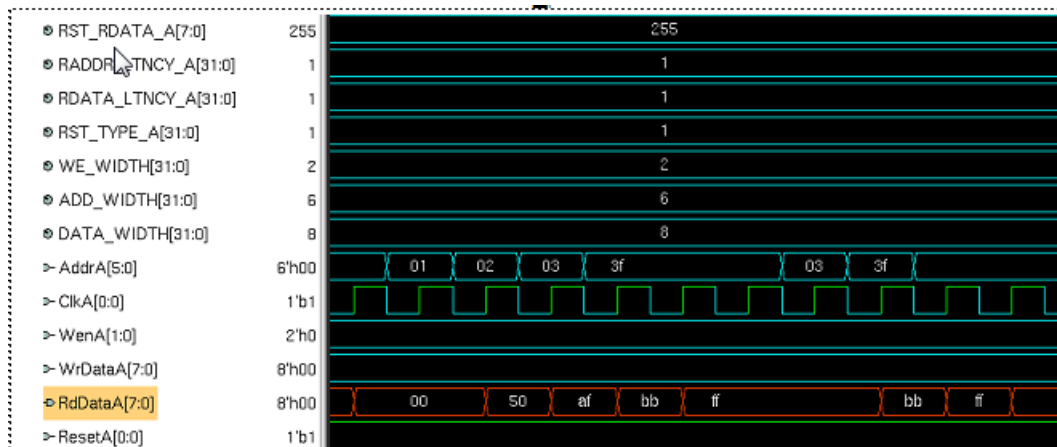
On each active edge of the clock when there is a change in address, data is valid on the same clock or next clock (depending on latency parameter values for read address and read data ports). Active reset ignores any change in input address, and data and output data are initialized to user-defined values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

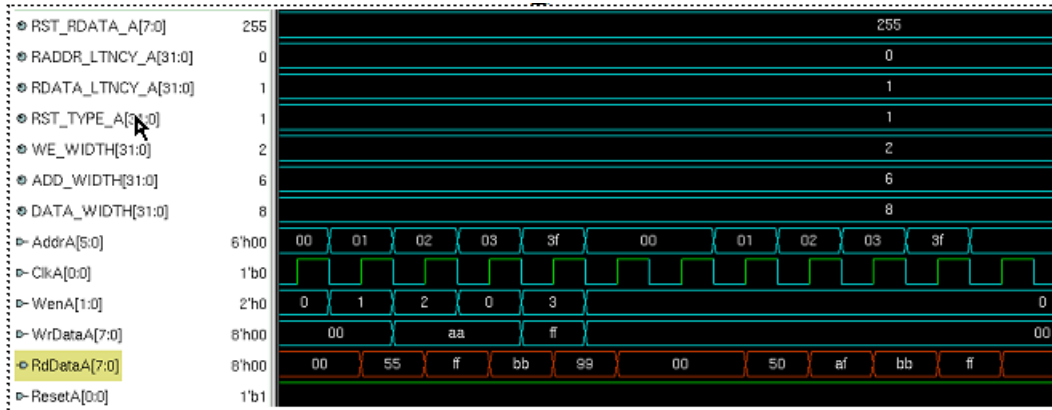


As shown in the above waveform, output read data changes on the next clock edge after the input address changes. When the address changes from 'h00 to 'h01, read data changes to 50 on the next clock, and data is valid on the next clock edge.

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:

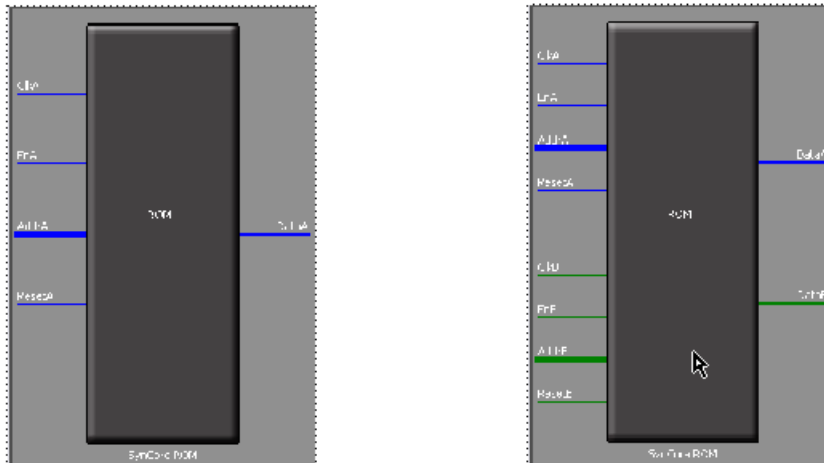
- [Functional Overview](#), on page 336
- [Single-Port Read Operation](#), on page 338
- [Dual-Port Read Operation](#), on page 338
- [Parameter List](#), on page 339
- [Clock Latency](#), on page 340

For further information, refer to the following:

- [Specifying ROMs with SYNCore](#), on page 470 of the *User Guide*, for information about using the wizard to generate ROMs
- [Launch SYNCore Command](#), on page 242 and [SYNCore ROM Wizard](#), on page 261 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.



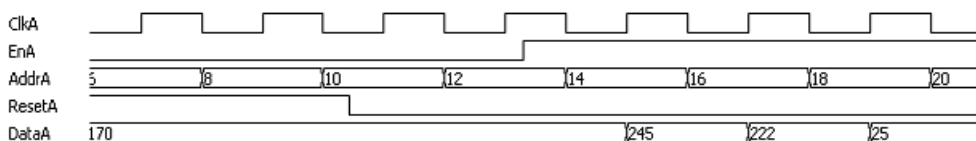
In the single-port (Port A) configuration, signals are synchronized to ClkA; ResetA can be synchronous or asynchronous depending on parameter selection. The read address (AddrA) and/or data output (DataA) can be registered to increase memory performance and improve timing. Both the read address and data output are subject to clock latency based on the ROM configuration (see [Clock Latency, on page 340](#)). 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 470](#) 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.

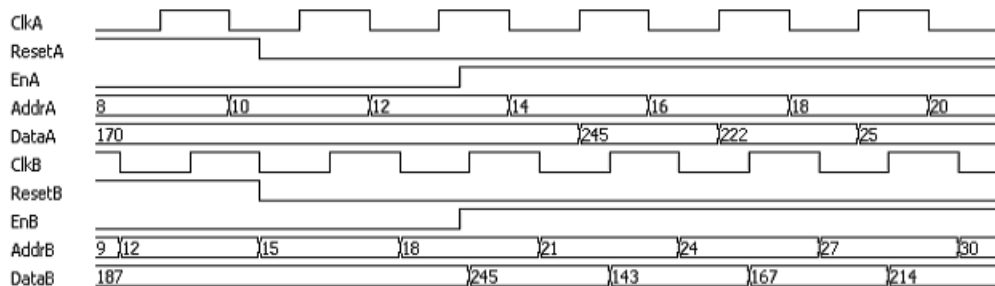


When reset is active, the output data holds the initialization value (i.e., 255). When reset goes inactive (and enable is active), data is read from the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset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

Note: In the above timing diagram, reset is synchronous. Clock latency varies according to the implementation and parameters as described in [Clock Latency, on page 340](#).

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 470](#) 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 from the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset are inactive, the output holds its previous read value.

Note: In the above timing diagram, reset is synchronous. Clock latency varies according to the implementation and parameters as described in [Clock Latency, on page 340](#).

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

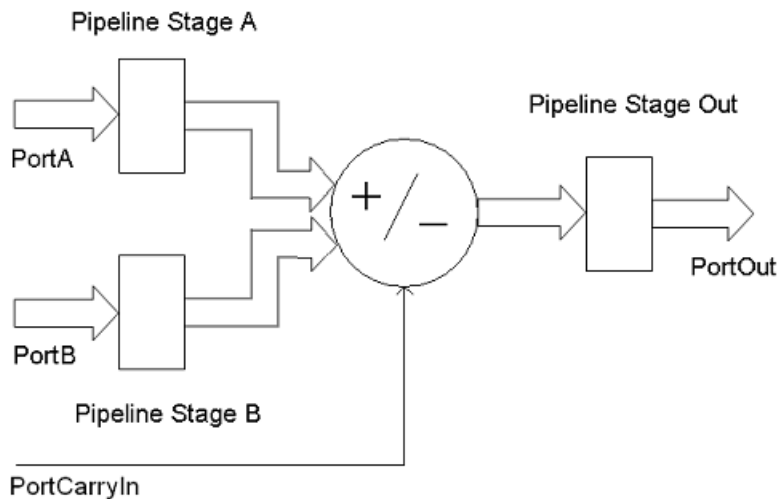
The SYNCore adder/subtractor compiler generates Verilog code for a parameterizable, pipelined adder/subtractor. This section describes the functionality of this block in detail.

Functional Description

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

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

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



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

- Adder
- Subtractor
- Dynamic Adder and Subtractor

Adder

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

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

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

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

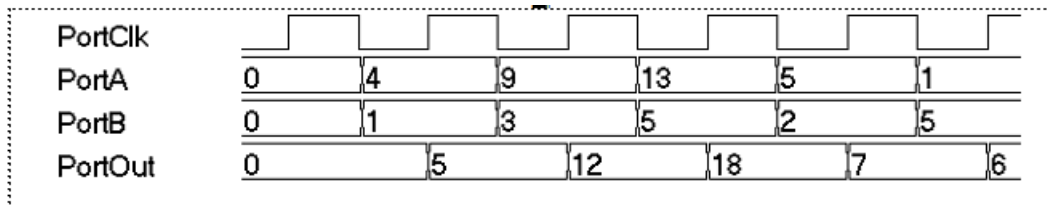
Adder with No Pipeline Stages – In this mode, the port A and port B inputs are added. The adder is purely combinational, and the output changes immediately with respect to the inputs.

Parameters: PORTA_PIPELINE_STAGE= '0'

PortA	0	4	9	13	5	1
PortB	0	1	3	5	2	5
PortOut	0	5	12	18	7	6

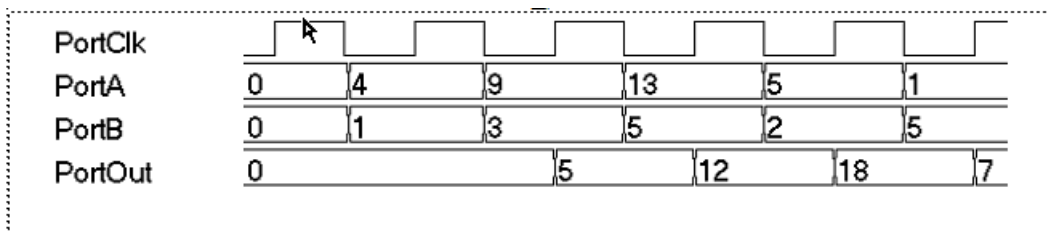
Adder with Pipeline Stages at Input Only – In this mode, the port A and port B inputs are pipelined and added. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

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 through the parameter `CONSTANT_VALUE`). Optional pipeline stages can also be inserted at port A. Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the adder configurations are given below (here `CONSTANT_VALUE= '3'`)

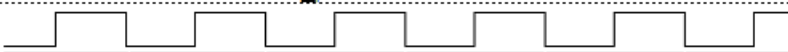
Adder with No Pipeline Stages – In this mode, input port A is added with a constant value. The adder is purely combinational, and the output changes immediately with respect to the input.

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

PortA	0	4	1	9	3	13
PortOut	3	7	4	12	6	16

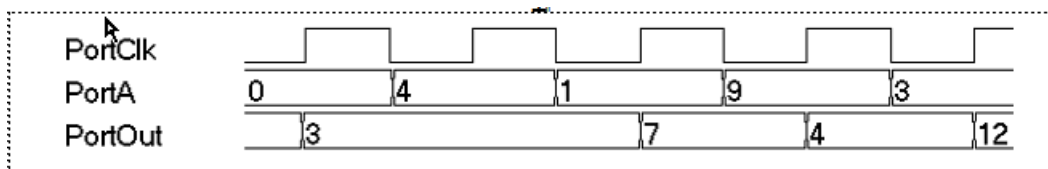
Adder with Pipeline Stage at Input Only – In this mode, input port A is pipelined and added with a constant value. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

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

PortClk						
PortA	0	4	1	9	3	13
PortOut	3	7	4	12	6	16

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

Parameters: PORTA_PIPELINE_STAGE= '1'
 PORTOUT_PIPELINE_STAGE= '1'



Subtractor

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

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

`CONSTANT_PORT='1'` – subtractor with one constant port

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

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

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

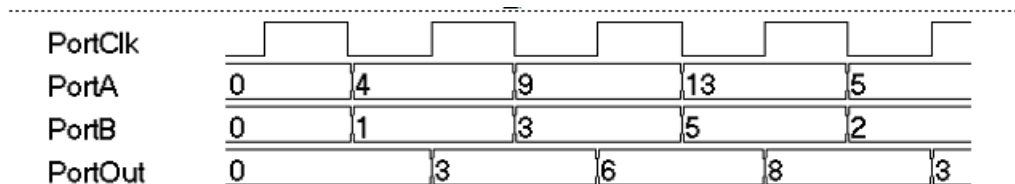
Parameters:

- `PORTA_PIPELINE_STAGE= '0'`
- `PORTB_PIPELINE_STAGE= '0'`
- `PORTOUT_PIPELINE_STAGE= '0'`

PortA	0	4	9	13	5
PortB	0	1	3	5	2
PortOut	0	3	6	8	3

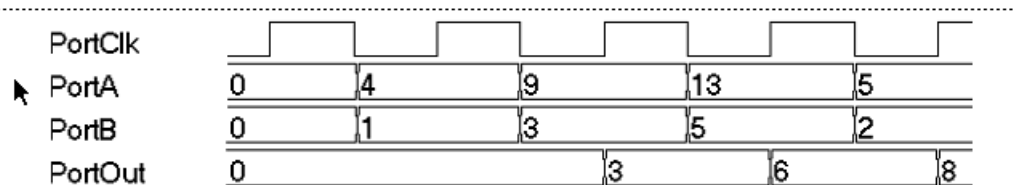
Subtractor with Pipeline Stages at Input Only – In this mode, input port B and input PortA are pipelined and then subtracted. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

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



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 through the parameter `CONSTANT_VALUE`). Optional pipeline stages can also be inserted at port A. Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, a number of the subtractor configurations are given below (here `CONSTANT_VALUE='1'`).


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

Parameters: `PORTA_PIPELINE_STAGE='0'`
 `PORTOUT_PIPELINE_STAGE='0'`

PortA	0	4	1	9	3
PortOut	0	3	0	8	2

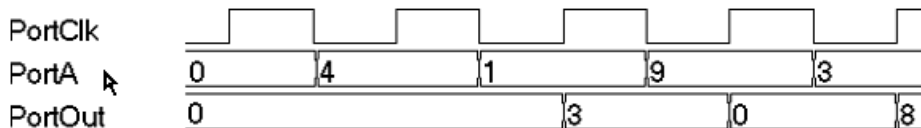
Subtractor with Pipeline Stages at Input Only – In this mode, a constant value is subtracted from pipelined input port A. Because there is no pipeline stage at the output, the output is valid at each rising edge of the clock.

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

PortClk					
PortA	0	4	1	9	3
PortOut	0	3	0	8	2

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

Parameters: PORTA_PIPELINE_STAGE= '1'
 PORTOUT_PIPELINE_STAGE= '1'



Dynamic Adder/Subtractor

In dynamic adder/subtractor mode, port PortADDnSUB controls adder/subtractor operation.

PortADDnSUB='0' – adder operation

PortADDnSUB='1' – subtractor operation

Based on the parameter CONSTANT_PORT the dynamic adder/subtractor can be configured in one of two ways:

CONSTANT_PORT='0' – dynamic adder/subtractor with two input ports

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

Dynamic Adder/Subtractor with Two Input Ports (Port A and Port B)

In this mode, the addition and subtraction is dynamic based on the value of input port PortADDnSUB. Optional pipeline stages can also be inserted at Port A, Port B, or both Port A and Port B. Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, some of the dynamic adder/subtractor configurations are given below.

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

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

PortADDnSUB				
PortA	5	15	8	13
PortB	7	2	5	
PortOut	12	17	13	8

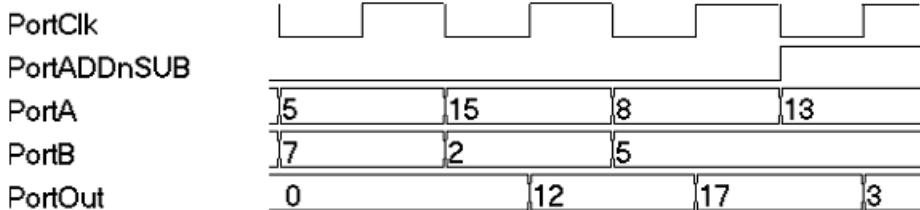
Dynamic Adder/Subtractor with Pipeline Stages at Input Only – In this mode, input port A and port B are pipelined and then added/subtracted based on the value of port PortADDnSUB. Because there is no pipeline stage at the output port, the result immediately changes with respect to the PortADDnSUB signal.

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

PortClk				
PortADDnSUB				
PortA	5	15	8	13
PortB	7	2	5	
PortOut	12	17	13	3
				8

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'

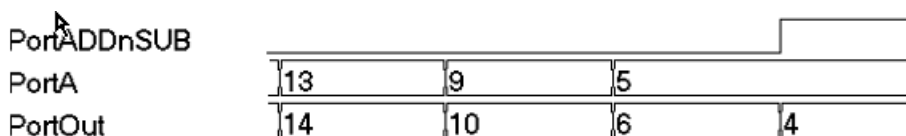


Dynamic Adder/Subtractor with a Port Constant

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').

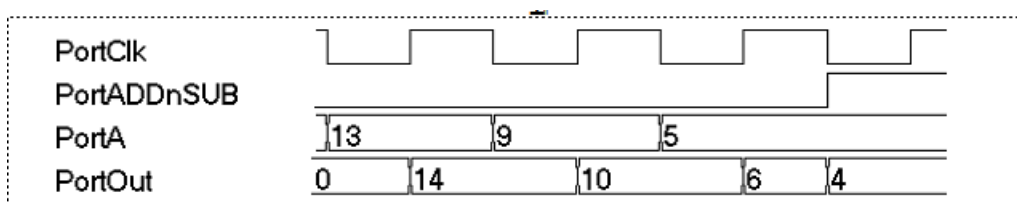
Dynamic Adder/Subtractor with No Pipeline Registers – In this mode, dynamic adder/subtractor is a purely combinational, and the output change immediately with respect to the input.

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



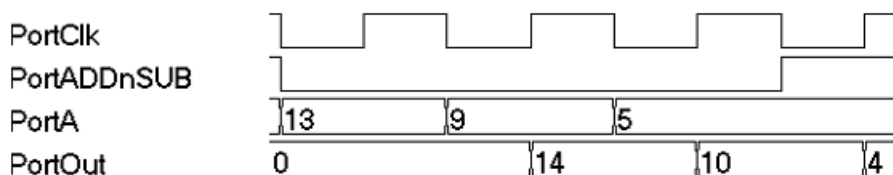
Dynamic Adder/Subtractor with Pipeline Stages at Input Only – In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because there is no pipeline stage on the output port, the result changes immediately with respect to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

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



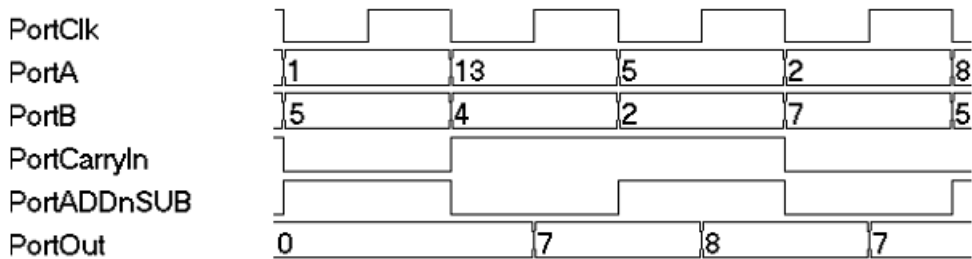
Dynamic Adder/Subtractor with Pipeline Stages at Input and Output – In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because the output port is pipelined, the result is valid only on the second rising edge of the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

Parameters: PORTA_PIPELINE_STAGE= '1'
 PORTOUT_PIPELINE_STAGE= '1'



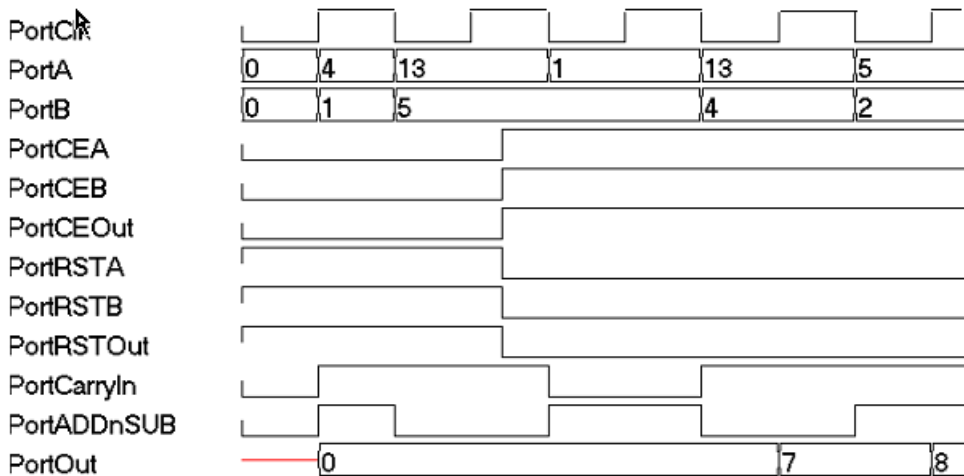
Dynamic Adder/Subtractor with Carry Input

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 354](#)
- [UP Counter Operation, on page 355](#)
- [Down Counter Operation, on page 355](#)
- [Dynamic Counter Operation, on page 356](#)

For further information, refer to the following:

- [Specifying Counters with SYNCore, on page 482](#) of the *User Guide*, for information about using the wizard to generate a counter core.
- [Launch SYNCore Command, on page 242](#) and [SYNCore Counter Wizard, on page 269](#) for descriptions of the interface and generating the core.

Functional Overview

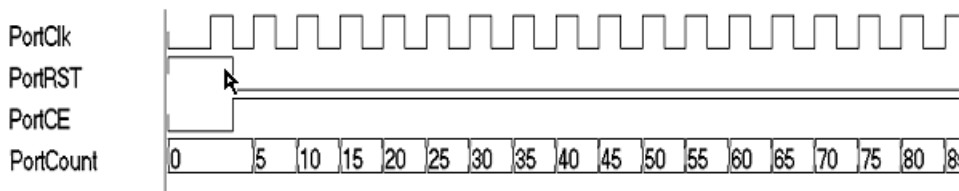
The SYNCore counter component supports up, down, and dynamic (up/down) counter implementations using DSP blocks or logic elements. For each configuration, design optimizations are made for optimum use of core resources.

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 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'

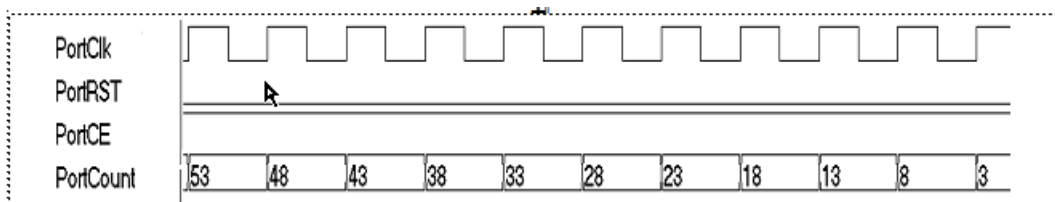


Note: Counter core can be configured to use a constant or dynamic load value in Up Counter mode (for the counter to load the PortLoadValue, PortCE must be active). This functionality is explained in [Dynamic Counter Operation, on page 356](#).

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 356](#).

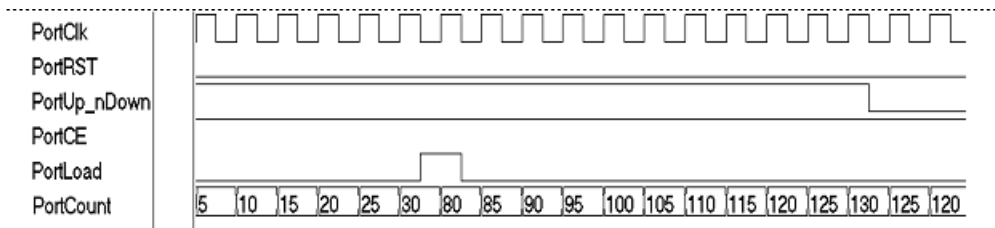
Dynamic Counter Operation

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'



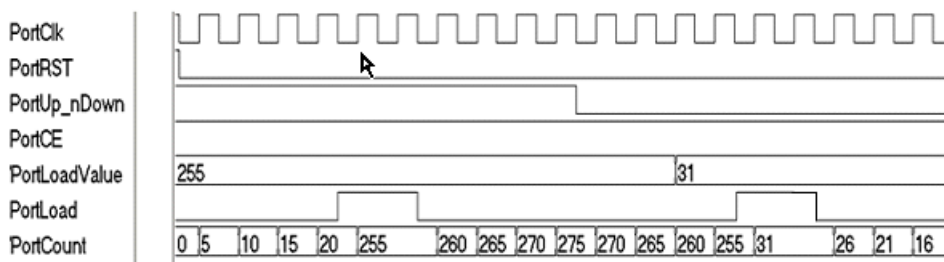
Note: *For counter to load the PortLoadValue, PortCE must be active.

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.

Encryption Scripts

There are two FPGA encryption methods available to Synopsys FPGA synthesis tool users: IEEE 1735-2014 and OpenIP. With both encryption methods, the IP vendor can encrypt their IP from their own website. From the synthesis tool, the synthesis user has access to the IP that the vendor makes available for download and evaluation within a synthesis design.

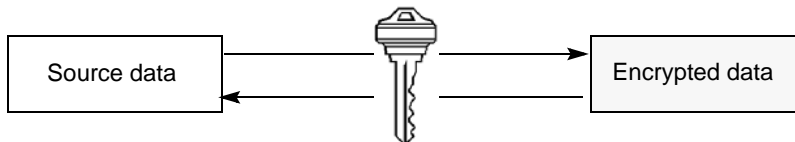
Each encryption method has corresponding scripts that the user can run. The following sections provide an overview of encryption and decryption methodologies and descriptions of the two encryption scripts:

- [Encryption and Decryption Methodologies, on page 359](#)
- [The encryptP1735 Script, on page 360](#) (for IEEE 1735-2014 encryption)
- [The encryptIP Script, on page 364](#) (for OpenIP encryption)

Encryption and Decryption Methodologies

This section describes common encryption schemes. There are two major classes of encryption/decryption algorithms: symmetric, and asymmetric.

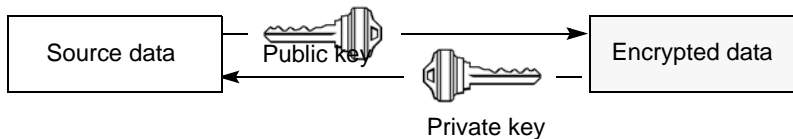
- **Symmetric Encryption**
With this kind of encryption, a special number is used as a key to encrypt the files. The same key is used to decrypt the file, so the software must have access to the same key.



The supported algorithms are:

- Data Encryption Standard (DES)
- Triple DES
- Advanced Encryption Standard (AES)
- **Asymmetric Encryption**
This encryption scheme uses different keys to encode and decode data. The EDA tool vendor generates the keys and makes a public key to

everyone who needs it for encryption. The public key cannot be used for decryption. The EDA tool uses the private key to decrypt the data. The asymmetric encryption cipher used is RSA.



The encryptP1735 Script

The encryptP1735 script is available to IP vendors who wish to provide IP to their synthesis users. The script is a Perl script that uses the IEEE 1735-2014 standard to let IP vendors encrypt modules or components, which can then be downloaded for evaluation or use by a Synopsys FPGA user. The script can be run according to any of three use models to encrypt the associated RTL files (see [Encrypting IP with the encryptP1735.pl Script](#), on [page 494](#) of the *User Guide*).

You run the script with the encryptP1735 command, the complete syntax for which is described in [encryptP1735](#), on [page 29](#) of the *Command Reference*.

The following sections describe details of the encryptP1735 script files:

- [Input Files for the IEEE 1735-2014 Encryption Script, on page 361](#)
- [Public Keys Repository File, on page 361](#)
- [Pragmas for the IEEE 1735-2014 Encryption Script, on page 361](#)
- [Adding Multiple Keys, on page 362](#)
- [Limitations, on page 364](#)

Input Files for the IEEE 1735-2014 Encryption Script

The encryptP1735 encryption script reads an HDL file, with or without encryption attributes, according to the selected use model. Additionally, the script reads the keys repository file that contains the public keys for the IP consumer tools.

Public Keys Repository File

The encryptP1735 encryption script requires public keys from the default keys.txt file from the directory *installLocation/lib* to create the decryption envelope. This file includes public keys for each of the tools that require a key block in the encrypted file. The public keys file includes a Synopsys synthesis tool public key; the file can be expanded by the user to include public keys for other tools.

Pragmas for the IEEE 1735-2014 Encryption Script

The header blocks in the encryptP1735.pl script support the pragmas described in the following table.

Pragma Keyword	Description
begin	Opens a new encryption envelope
end	Closes an encryption envelope
begin_protected	Opens a new decryption envelope
end_protected	Closes a decryption envelope
author	Identifies the author of an envelope
author_info	Specifies additional author information

Pragma Keyword	Description
encoding	Specifies the coding scheme for the encrypted data
data_keyowner	Identifies the owner of the data encryption key
data_method	Identifies the data encryption algorithm
data_keyname	Specifies the name of the data encryption key
data_block	Begins an encoded block of encrypted data
encrypt_agent	Identifies the encryption service
encrypt_agent_info	Specifies additional encryption-agent information
key_keyowner	Identifies the owner of the key encryption key
key_method	Specifies the key encryption algorithm
key_keyname	Specifies the name of the key encryption key
key_public_key	Specifies the public key for key encryption
key_block	Begins an encoded block of key data
version	P1735 encryption version
comment	Uninterrupted documentation string

Adding Multiple Keys

It may become necessary to add multiple keys to the RTL to support how multiple vendors access to the same RTL. Multiple vendor access is done by editing the Synopsys synthesis tool public key included in the install/lib/keys.txt key file shown below:

```
// Use verilog pragma syntax in this file

`pragma protect version=1
`pragma protect author="default"
`pragma protect author_info="default"

`pragma protect key_keyowner="Synopsys", key_keyname="SYNP05_001", key_method="rsa"
`pragma protect key_public_key

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAybsQaMidichZyh14wbXn
UpP8lK+jJY5oLpGqDfSW5PMXBVp0Wfd1d32onXEpRkwxEJLlK4RgS43d0FG2ZQ1l
irdimRKNnUtPxsrJzbMr74MQkwmg/X7SEe/1EqwK9Uk77cMEncLycI5yX4f/K9Q9
W55nLD+Nh6BL7kwR0vSevfePC1fkOaluC7b7Mwb1mcqCLBRRP9/eF0wUIoxVRzjA
+pJvORwhYtZEhnwvTb1BJsnynet1LfDi/D5WZoikTP/0KBiP87QHMSuVBydMA7J7
g6sxKB92hx2Dpv1ojds1Y5ywjxFxOAA93nFjmLsJq3i/P0lv5TmtncYX3Wkryw4B
eQIDAQAB
```

```
// Add additional public keys below this line
// Add additional public keys above this line

`pragma protect data_keyowner="default-ip-vendor"
`pragma protect data_keyname="default-ip-key"
`pragma protect data_method="aes128-cbc"

// End of file
```

The file is expanded to include public keys for other tools. Please add any other key between the lines:

```
// Add additional public keys below this line

// Add additional public keys above this line
```

The following is an example of an expanded public keys file that contains dummy keys with key_keyname = "DUMMY":

```
// Use verilog pragma syntax in this file

`pragma protect version=1
`pragma protect author="default"
`pragma protect author_info="default"

`pragma protect key_keyowner="Synopsys", key_keyname="SYNP05_001", key_method="rsa"
`pragma protect key_public_key

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAYbsQaMidichZyh14wbXn
UpP8lK+jJY5oLpGqDfSW5PMXBVp0WFd1d32onXEpRkwxEJLlK4RgS43d0FG2ZQ1l
irdimRKNnUtPxsrJzbMr74MQkwmG/X7SEe/1EqwK9Uk77cMEncLycI5yX4f/K9Q9
WS5nLD+Nh6BL7kwR0vSevfePClfkOaluC7b7MwblmcqCLBRRP9/eF0wUIoxVRzjA
+pJvORwhYtZEhnwvTb1BJsnyneT1LfDi/D5WZoikTP/0KBiP87QHMSuVByDMA7J7
g6sxKB92hx2Dpv1ojds1Y5ywJxFxOAA93nFjmLsJq3i/P0lv5TmtnCXY3Wkryw4B
eQIDAQAB

// Add additional public keys below this line

`pragma protect key_keyowner="Synopsys", key_keyname="DUMMY", key_method="rsa"
`pragma protect key_public_key

.....

// Add additional public keys above this line

`pragma protect data_keyowner="default-ip-vendor"
`pragma protect data_keyname="default-ip-key"
`pragma protect data_method="aes128-cbc"

// End of file
```

If you are using a partial file with all pragmas use model, the keyowner entries also must be edited:

```
`protect key_keyowner="Synopsys", key_keyname="SYNP05_001", key_method="rsa", key_block
`protect key_keyowner=" Synopsys", key_keyname=" DUMMY", key_method="rsa", key_block
`protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip", data_method="des-cbc"
```

Be sure to include the below and above entries within your RTL to be able to generate the decryption envelope properly. It is very important that you follow this process when using the partial file with all pragmas use model. If you are using the full-file use model or partial file with minimal pragmas use model, editing the RTL is unnecessary when adding additional keys to the default public key to create expanded keys.

Note: The expanded key shown is only an example and is not intended be used with the encryptP1735.pl script; the actual key must be obtained from a valid EDA vendor and added to the keys.txt file.

Limitations

To use the encryptP1735.pl script on Windows, Perl must be installed. Several commercial and free versions are available including from <http://www.perl.org/get.html>.

However, the script works correctly on a Linux 64-bit platform.

The encryptIP Script

The encryptIP Perl script is a Synopsys FPGA IP script that is provided to IP vendors who wish to provide IP to synthesis users. The script uses the OpenIP scheme to encrypt modules or components, which can then be downloaded for evaluation or use by the Synopsys FPGA user. You can download the encryptIP Perl script from SolvNet (<https://solvnet.synopsys.com/retrieve/032343.html>).

You run the script with the encryptIP command, the complete syntax for which is described in [encryptIP](#), on page 25 of the *Command Reference*.

For details, see the following:

- [The encryptIP Script Run](#), on page 365
- [Pragmas Used in the encryptIP Script](#), on page 367

The encryptIP Script Run

The following example describes the various steps that the encryptIP script executes. For descriptions of the pragmas used in the encryptIP script, see [Pragmas Used in the encryptIP Script, on page 367](#).

1. For each RTL file, the script creates a data block using symmetric algorithm and your own session key.

You can use any of the CBC encryption modes listed in [encryptIP, on page 25](#) of the *Command Reference* manual. The initialization vector is a constant, and the block is encoded in base 64. The following excerpt uses the data encryption key ABCDEFG:

```
%% protect data_block
UWhcm3CPmGz27DXAWQZF8rY7hSsvLwedXiP59HYZHJfoMIMkJ0W+6H7vmJEXZ/
dTxnAgGB2YJCF7lsaZ6x6kRisdtBtIo8+0Glskcykt7FtpAjpz24cJ9hoSYMu
HCmG70dHDNzTHWjkwBs2fxo5S6559d3pW+SDutrsvrsHntyvHYiqxUZPsGce
ZZQJpqQIpqo24uFCVHNSX/URvL47CUBWoKB2XEpyRv5Zgd1F52YjBLIpET
+kBEzutAorF5rD9eZSALS0kvVb7MPXFxmFCF8wHwTnRtkPthNCMq0t3iCgf9EH
```

2. The script precedes the data block with a small data block header that describes the data method:

```
%% protect data_method=aes128-cbc
%% protect data_block
UWhcm3CPmGz27DXAWQZF8rY7hSsvLwedXiP59HYZHJfoMIMkJ0W+6H7vmJEXZ/
dTxnAgGB2YJCF7lsaZ6x6kRisdtBtIo8+0Glskcykt7FtpAjpz24cJ9hoSYMu
HCmG70dHDNzTHWjkwBs2fxo5S6559d3pW+SDutrsvrsHntyvHYiqxUZPsGce
ZZQJpqQIpqo24uFCVHNSX/URvL47CUBWoKB2XEpyRv5Zgd1F52YjBLIpET
+kBEzutAorF5rD9eZSALS0kvVb7MPXFxmFCF8wHwTnRtkPthNCMq0t3iCgf9EH
```

3. The script then prepares a key block for the Synplify tool, which contains your session key and Synopsys-specific directives. Note that the `data_decrypt_key` is required.

```
output_method=blackbox (Your IP will be a black box in the output netlist)
data_decrypt_key=ABCDEFG (Session key you used to encrypt your data block)
```

See [encryptIP, on page 25](#) in the *Command Reference* manual for information about output methods.

4. Use Synplify Public key (Synopsys has an executable that returns this key) and RSA2048 asymmetric encryption to create a key block. Encode it in base64.

```

%%% protect key_block
U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJDfonqfqzjzEQ+xQ4
wyh65wo6X56Jm+ClavUzjgQKK0c4y47nyAlIWcuq1Nh6KeuUscxp+nL6yT9Am
+nv+c57jSCMG0QsFbRBAIhdlohQAbYbSIuFxdLFEFw4znF3+YDAsMHeIs
ltqxKqhQzYQ2fGJdQz0NVRilhfjx/RpGmoXmzvSTX2xsre+ZNDh3r9qvj37
QGwLH2erPt/iXcUVnlPCoAV5z8M1YLrKY8ui7KBs/HhyP7L2mAMPQAFY3i
DhycUcJ5sirBgKZycpkhP8jQ02yjTZMb7z9KyYTHrzDdA==

%%% protect key_block
+700tKshqQgTmb8UdRU9n263KwF7RWb8GSz7Cwo6X56Jm+ClxISekIJDfon
qfqzjzEQ+xQ4wyh657nyAlIWcuq1Nh6KeuUscxp+nL6yT9AaVuZjG0QsFb
RBAIhdlohQAbYbgQKK0c4y4m+nv+c57jSCMxW4znF3+YDAsMHeIsltqx
KqhQzYQ2fGJdQz0NXmzvSTX2xsre+ZNVRIlhFjSIuFxdLFEFw4znF3+YDAsMHeIs
37QGwLH2erPt/iXcUVnlPCO7KBs/HhyP7L2mAMPQAFY3iDhycUcJaV5z8MD
h3r1YLrKY8ui8jQ02yjTZMb7z9pkhPYTHrzDdAKy5sirBgKZyc==

```

5. The script adds a small key block header to each key block.

```

%%% protect key_keyowner=Synopsys
%%% protect key_keyname=SYNP05_001
%%% protect key_block
U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJDfonqfqzjzEQ+xQ4

```

6. Your final encrypted IP key and data blocks looks like this:

```

%%% protect protected_file 1.0
<optional unencrypted HDL>
%%% protect begin_protected
%%% protect key_keyowner=Synopsys
%%% protect key_keyname=SYNP05_001

%%% protect key_block
U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJDfonqfqzjzEQ+xQ4
wyh65wo6X56Jm+ClavUzjgQKK0c4y47nyAlIWcuq1Nh6KeuUscxp+nL6yT9Am
+nv+c57jSCMG0QsFbRBAIhdlohQAbYbSIuFxdLFEFw4znF3+YDAsMHeIs
ltqxKqhQzYQ2fGJdQz0NVRilhfjx/RpGmoXmzvSTX2xsre+ZNDh3r9qvj37
QGwLH2erPt/iXcUVnlPCoAV5z8M1YLrKY8ui7KBs/HhyP7L2mAMPQAFY3i
DhycUcJ5sirBgKZycpkhP8jQ02yjTZMb7z9KyYTHrzDdA==

<other key blocks>

%%% protect data_method=aes128-cbc
%%% protect data_block
UWhcm3CPmGz27DXAWQZF8ry7hSsvLwedXiP59HYZHJfoMIMkJ0W+6H7vmJEXZ/
dTxnAgGB2YJCF7lsaZ6x6kRisdtBtIo8+0Gls/kcykt7FtpAjzp24cJ9ho
SYMuhCmG70dHDNzTHWjkwBs2fxo5S6559d3pW+SDutrvrshntyv
HYiqxUZPsGceZZQJpqQIpqo24uFCVHNSX/URvL47CUBWoKB2XEPyRv5Zg
...

```

```
%%% protect end_protected
<optional unencrypted HDL>
```

Pragmas Used in the encryptIP Script

The header blocks in the encryptIP script use the pragmas described in the following tables. Note the following:

- The %%% protect directive must be placed at the exact beginning of a line.
- Exactly one white-space character must separate the %%% from the command that follows

The following table describes the general pragmas used:

%%% protect protected_file 1.0	Line 1 of protected file
%%% protect begin_protected	Begin protected section
%%% protect end_protected	Ends protected section
%%% protect comment <i>comment</i>	Single line plain-text comment
%%% protect begin_comment	Begin block of plain-text comments
%%% protect end_comment	End block of plain-text block comments

The following table describes the data block pragmas:

%%% protect author= <i>string</i>	Arbitrary string
%%% protect data_method= <i>string</i>	For all supported FPGA vendors, one of the DES encryption methods described in The encryptP1735 Script, on page 360 .
%%% protect data_block	Immediately precedes encrypted data block

The following table describes the key block pragmas:

%%% protect key_keyowner= <i>string</i>	Arbitrary string
---	------------------

%% protect key_keyname= <i>string</i>	Name recognized by the Synplify software to select key block
%% protect key_method= <i>string</i>	Encryption algorithm. Currently we support "rsa"
w %% protect key_block	Immediately precedes encrypted data block

CHAPTER 10

Scripts

This chapter describes Tcl scripts.

- *synhooks* [File Syntax, on page 370](#)
- [Tcl Script Examples, on page 372](#)

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 529](#).

Tcl Callback Syntax	Function
proc syn_on_set_project_template <i>{projectPath} {yourDefaultProjectSettings}</i>	Called when creating a new project. <i>projectPath</i> is the path name to the project being created.
proc syn_on_new_project <i>{projectPath}</i> <i>{yourCode}</i>	Called when creating a new project. <i>projectPath</i> is the path name to the project being created.
proc syn_on_open_project <i>{projectPath}</i> <i>{yourCode}</i>	Called when opening a project. <i>projectPath</i> is the path name to the project being created.
proc syn_on_close_project <i>{projectPath}</i> <i>{yourCode}</i>	Called after closing a project. <i>projectPath</i> is the path name to the project being created.
proc syn_on_start_application <i>{applicationName version currentDirectory}</i> <i>{yourCode}</i>	Called when starting the application. <ul style="list-style-type: none"> <i>applicationName</i> is the name of the software. For example synplyfy_pro. <i>version</i> is the name of the version of the software. For example 8.4 <i>currentDirectory</i> is the name of the software installation directory. For example C:\synplyfy_pro\bin\synplyfy_pro.exe.
proc syn_on_exit_application <i>{applicationName version}</i> <i>{yourCode}</i>	Called when exiting the application. <ul style="list-style-type: none"> <i>applicationName</i> is the name of the software. For example synplyfy_pro. <i>version</i> 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_F11 {}
{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 372](#)
- [Different Clock Frequency Goals, on page 372](#)
- [Setting Options and Timing Constraints, on page 373](#)

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 {

    ProASIC3 ProASIC3E Fusion # 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 PROASIC3E
```

```

set_option -part A2F200M3F
set_option -package PQFP208
set_option -speed_grade -2

# 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 Microsemi

The following topics describe how to design and synthesize with the Microsemi technology:

- [Basic Support for Microsemi Designs, on page 378](#)
- [Microsemi Components, on page 381](#)
- [Output Files and Forward-annotation for Microsemi, on page 407](#)
- [Optimizations for Microsemi Designs, on page 410](#)
- [Integration with Microsemi Tools and Flows, on page 419](#)
- [Microsemi Device Mapping Options, on page 421](#)
- [Microsemi Tcl set_option Command Options, on page 423](#)
- [Microsemi Attribute and Directive Summary, on page 426](#)

Basic Support for Microsemi Designs

This section describes the use of the tool with Microsemi devices. It describes

- [Microsemi Device-specific Support, on page 378](#)
- [Microsemi Features, on page 379](#)
- [Synthesis Constraints and Attributes for Microsemi, on page 379](#)

Microsemi Device-specific Support

The tool creates technology-specific netlists for a number of Microsemi families of FPGAs. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Implementation Options dialog box (see [Device Panel, on page 205](#)).

The following technologies are supported:

FPGAs	Technology Families
Mixed-Signal	<ul style="list-style-type: none">• SmartFusion2 and SmartFusion• Fusion
Low-Power	<ul style="list-style-type: none">• IGLOO Series (IGLOO2, IGLOOE, IGLOO+, and IGLOO)• ProASIC3 Series (ProASIC3L, ProASIC3E, and ProASIC3)
Rad-Tolerant FPGAs	RTG4, RT ProASIC3

After synthesis, the tool generates EDIF netlists as well as a constraint file that is forward annotated as input into the Microsemi Libero tool.

Microsemi Features

The synthesis tool contains the following Microsemi-specific features:

- Direct mapping to Microsemi c-modules and s-modules
- Timing-driven mapping, replication, and buffering
- Inference of counters, adders, and subtractors; module generation
- Automatic use of clock buffers for clocks and reset signals
- Automatic I/O insertion. See [I/O Insertion, on page 412](#) for more information.

Synthesis Constraints and Attributes for Microsemi

The synthesis tools let you specify timing constraints, general HDL attributes, and Microsemi-specific attributes to improve your design. You can manage the attributes and constraints in the SCOPE interface. Microsemi has vendor-specific I/O standard constraints it supports for synthesis. For a list of supported I/O standards, see [Microsemi I/O Standards, on page 379](#).

Microsemi I/O Standards

The following table lists the supported I/O standards for the ProASIC3L, ProASIC3E, Fusion, IGLOOe ProASIC3, IGLOO, and IGLOO+ families. Some I/O standards have associated modifiers you can set, such as slew, termination, drive, power, and Schmitt, which allow the software to infer the correct buffer types.

ProASIC3L, ProASIC3E, IGLOOe, Fusion	IGLOO, IGLOO+, ProASIC3
GTL25	LVC MOS_12
GTL+25	LVC MOS_15
GTL33	LVC MOS_18
GTL+33	LVC MOS_33
HSTL_Class_I	LVC MOS_5
HSTL_Class_II	LVDS

ProASIC3L, ProASIC3E, IGLOOe, Fusion	IGLOO, IGLOO+, ProASIC3
---	------------------------------------

LVC MOS_12	LVPECL
------------	--------

LVC MOS_15	LVTTTL
------------	--------

LVC MOS_18	PCI
------------	-----

LVC MOS_33	PCIX
------------	------

LVC MOS_5	
-----------	--

LVDS	
------	--

LVPECL	
--------	--

LVTTTL	
--------	--

PCI	
-----	--

PCIX	
------	--

SSTL_2_Class_I	
----------------	--

SSTL_2_Class_II	
-----------------	--

SSTL_3_Class_I	
----------------	--

SSTL_3_Class_II	
-----------------	--

See Also:

- [Industry I/O Standards, on page 178](#) for a list of industry I/O standards.
- [Microsemi Attribute and Directive Summary, on page 426](#) for a list of Microsemi attributes and directives.

Microsemi Components

The following topics describe how the synthesis tools handle various Microsemi components, and show you how to work with or manipulate them during synthesis to get the results you need:

- [Macros and Black Boxes in Microsemi Designs, on page 381](#)
- [DSP Block Inference, on page 383](#)
- [Microsemi RAM Implementations, on page 387](#)
- [Instantiating RAMs with SYNCORE, on page 405](#)
- [Control Signals Extraction for Registers \(SLE\), on page 406](#)

Macros and Black Boxes in Microsemi Designs

You can instantiate Smartgen¹ macros or other Microsemi macros like gates, counters, flip-flops, or I/Os by using the supplied Microsemi macro libraries to pre-define the Microsemi macro black boxes. For certain technologies, the following macros are also supported:

- [MACC and RAM Timing Models](#)
- [SIMBUF Macro](#)
- [MATH18X18 Block](#)
- [Microsemi Fusion Analog Blocks](#)
- [SmartFusion Macros](#)
- [SmartFusion2 MACC Block](#)

1. Smartgen macros now replace the ACTgen macros. ACTgen macros were available in the previous Designer 6.x place-and-route tool.

For general information on instantiating black boxes, see [Instantiating Black Boxes in VHDL, on page 300](#), and [Instantiating Black Boxes in Verilog, on page 90](#). For specific procedures about instantiating macros and black boxes and using Microsemi black boxes, see the following sections in the *User Guide*:

- [Defining Black Boxes for Synthesis, on page 348](#)
- [Using Predefined Microsemi Black Boxes, on page 540](#)
- [Using Smartgen Macros, on page 541](#)

MACC and RAM Timing Models

MACC and RAM timing models are supported in SmartFusion2 and IGLOO2. Timing analysis considers the timing arcs for RAM and MACC.

SIMBUF Macro

The synthesis software supports instantiation of the SIMBUF macro. The SIMBUF macro provides the flexibility to probe signals without using physical locations, such as possible from the Identify tool. The Resource Summary will report the number of SIMBUF instantiations in the IO Tile section of the log file.

SIMBUF macros are supported for ProASIC3, ProASIC3E, ProASIC3L, IGLOO, IGLOOe, IGLOO+, SmartFusion, and Fusion devices.

MATH18X18 Block

The synthesis software supports instantiation of the MATH18X18 block. The MATH18X18 block is useful for mapping arithmetic functions.

Microsemi Fusion Analog Blocks

Microsemi Fusion has several analog blocks built into the ProASIC3/3E device. The synthesis tool treats them as black boxes. The following is a list of the available analog blocks.

- AB
- NVM
- XTLOSC

- RCOSC
- CLKSRC
- NGMUX
- VRPSM
- INBUF_A
- INBUF_DA
- OUTBUF_A
- CLKDIVDLY
- CLKDIVDLY1

SmartFusion Macros

The synthesis software supports the following SmartFusion macros:

- FAB_CCC
- FAB_CCC_DYN

SmartFusion2 MACC Block

SmartFusion2 devices support bit-signed 18x18 multiply-accumulate blocks. This architecture provides dedicated components called SmartFusion2 MACC blocks, for which DSP-related operations can be performed like multiplication followed by addition, multiplication followed by subtraction, and multiplication with accumulate. For more information, see [DSP Block Inference, on page 383](#).

DSP Block Inference

This feature allows the synthesis tools to infer DSP or MATH18x18 blocks for SmartFusion2 devices. The following structures are supported:

- DOTP Support

MACC block, when configured in DOTP mode, has two independent signed 9-bit x 9-bit multipliers followed by addition. The sum of the dual independent 9x9 multiplier (DOTP) result is stored in the upper 35 bits

of the 44-bit output. In DOTP mode, the MACC block implements the following equation:

$$P = D + (\text{CARRYIN} + C) + 512 * ((AL * BH) + (AH * BL)), \text{ when SUB} = 0$$

$$P = D + (\text{CARRYIN} + C) - 512 * ((AL * BH) + (AH * BL)), \text{ when SUB} = 1$$

Below is an example RTL which infers MACC block in DOTP mode after synthesis:

```
module dotp_add_unsign_syn ( ina, inb, inc, ind, ine, dout);

parameter widtha = 6;
parameter widthb = 7;
parameter widthc = 7;
parameter widthd = 8;
parameter widthe = 30;
parameter width_out = 44;

input [widtha-1:0] ina;
input [widthb-1:0] inb;
input [widthc-1:0] inc;
input [widthd-1:0] ind;
input [widthe-1:0] ine;
output reg [width_out-1:0] dout;

always @(ina or inb or inc or ind or ine) begin
    dout    <= (ina * inb) + (inc * ind) + ine ;
end

endmodule
```

MACC block does not support DOTP mode when:

- Width of the multiplier inputs is greater than 9-bits for signed.
- Width of the multiplier inputs is greater than 8-bits for unsigned.
- Width of the non-multiplier inputs is greater than 36-bits.
- Multipliers
- Mult-adds — Multiplier followed by an Adder
- Mult-subs — Multiplier followed by a Subtractor
- Wide multiplier inference

A multiplier is treated as wide, if any of its inputs is larger than 18 bits signed or 17 bits unsigned. The multiplier can be configured with only one input that is wide, or else both inputs are wide. Depending on the number of wide inputs for signed or unsigned multipliers, the synthesis software uses the cascade feature to determine how many math blocks to use and the number of Shift functions it needs.

- MATH block inferencing across hierarchy

This enhancement to MATH block inferencing allows packing input registers, output registers, and any adders or subtractors into different hierarchies. This helps to improve QoR by packing logic more efficiently into MATH blocks.

By default, the synthesis software maps the multiplier to DSP blocks if all inputs to the multiplier are more than 2-bits wide; otherwise, the multiplier is mapped to logic. You can override this default behavior using the `syn_multstyle` attribute. See [syn_multstyle, on page 121](#) for details.

The following conditions also apply:

- Signed and unsigned multiplier inferencing is supported.
- Registers at inputs and outputs of multiplier/multiplier-adder/multiplier-subtractor are packed into DSP blocks.
- Synthesis software fractures multipliers larger than 18X18 (signed) and 17X17 (unsigned) into smaller multipliers and packs them into DSP blocks.
- When multadd/multsub are fractured, the final adder/subtractor are packed into logic.

DSP Cascade Chain Inference

The MATH18x18 block cascade feature supports the implementation of multi-input Mult-Add/Sub for devices with MATH blocks. The software packs logic into MATH blocks efficiently using hard-wired cascade paths, which improves the QoR for the design.

Prerequisites include the following requirements:

- The input size for multipliers is *not* greater than 18x18 bits (signed) and 17x17 bits (unsigned).
- Signed multipliers have the proper sign-extension.

- All multiplier output bits feed the adder.
- Multiplier inputs and outputs can be registered or not.

Multiplier-Accumulators (MACC) Inference

The Multiplier-Accumulator structures use internal paths for adder feedback loops inside the MATH18x18 block instead of connecting it externally.

Prerequisites include the following requirements:

- The input size for multipliers is *not* greater than 18x18 bits (signed) and 17x17 bits (unsigned).
- Signed multipliers have the proper sign-extension.
- All multiplier output bits feed the adder.
- The output of the adder must be registered.
- The registered output of the adder feeds back to the adder for accumulation.
- Since the Microsemi MATH block contains one multiplier, only Multiplier-Accumulator structures with one multiplier can be packed inside the MATH block.

The other Multiplier-Accumulator structure supported is with Synchronous Loadable Register.

Prerequisites include the following requirements:

- All the requirements mentioned above apply for this structure as well.
- For the Loading Multiplier-Accumulator structure, new Load data should be passed to input C.
- The LoadEn signal should be registered.

DSP Limitations

Currently, DSP inferencing does not support the following functions:

- Overflow extraction
- Arithmetic right shift for operand C

Note: For more information about Microsemi DSP math blocks along with a comprehensive set of examples, see the *Inferring Microsemi RTAX-DSP MATH Blocks* application note on SolvNet.

Microsemi RAM Implementations

Refer to the following topics for Microsemi RAM implementations:

- [RAM64x18, RAM64x18_RT, RAM1K18_RT Enhancements, on page 387](#)
- [URAM Inference for Sequential Shift Registers, on page 387](#)
- [RAM Primitives Support in RTG4](#)
- [RAM Inference for SmartFusion2/IGLOO2/RTG4](#)
- [RAM Inference Enhancement for RTG4 \(ECC Support\)](#)
- [RAM Read Enable Extraction](#)
- [ProASIC3/3E/3L and IGLOO+/IGLOO/IGLOOe](#)
- [SmartFusion2](#)

RAM64x18, RAM64x18_RT, RAM1K18_RT Enhancements

Packing of enable signal on the read address register into RAM1K18_RT (A_REN), RAM64x18 (A_ADDR_EN & B_ADDR_EN), and RAM64x18_RT (A_ADDR_EN & B_ADDR_EN) is supported.

URAM Inference for Sequential Shift Registers

URAM inference for sequential shift registers is supported for SmartFusion2, IGLOO2, and RTG4 technologies.

By default, seqshift is implemented using registers. The `syn_srlstyle` attribute is used to override the default behavior of seqshift implementation using URAM.

The attribute can be applied on the top level module or on a seqshift instance in the RTL view, by dragging and dropping the instance to the SCOPE editor.

If the attribute is applied on the top level module, the tool infers URAM for all the seqshifts in the design, using the following threshold values:

Depth ≥ 8 and Depth*Width > 84

If the attribute is applied on the seqshift instance, the tool infers URAM irrespective of the threshold values.

syn_srlstyle Values

Value	Description
Registers	seqshifts are inferred as registers.
URAM	seqshift is inferred as: <ul style="list-style-type: none"> • RAM64x18 for SmartFusion2 and IGLOO2 devices • RAM64x18_RT for RTG4 device

syn_srlstyle Syntax

FDC	define_attribute {object} syn_srlstyle {registers uram } define_global_attribute syn_srlstyle {registers uram }
Verilog	object /* synthesis syn_srlstyle = "registers uram " */;
VHDL	attribute syn_srlstyle : string; attribute syn_srlstyle of object : signal is "registers uram ";

Example

The tool infers a seqshift primitive for the given RTL:

```
module p_seqshift(clk, we, din, dout) ;

parameter SRL_WIDTH = 7;
parameter SRL_DEPTH = 37;

input clk, we;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0]/*synthesis srlstyle =
"uram"*/;
integer i;
```

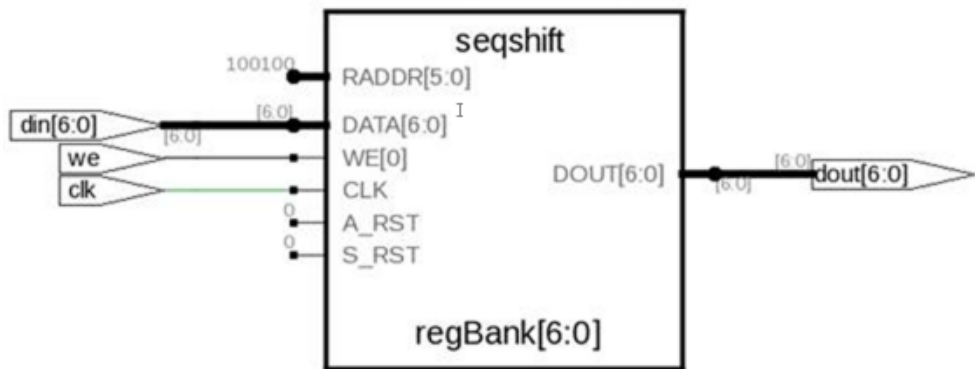
```

always @(posedge clk) begin
  if (we) begin
    for (i=SRL_DEPTH-1; i>0; i=i-1) begin
      regBank[i] <= regBank[i-1];
    end
    regBank[0] <= din;
  end
end

assign dout = regBank[SRL_DEPTH-1];
endmodule

```

The following graphic displays seqshift for the RTL above in technology view.



Limitations

- URAM inference for seqshift is not supported, if the output is taken from a dynamic stage.
- Seqshifts with synchronous reset or asynchronous reset are inferred as registers.
- Seqshifts with both synchronous reset and asynchronous reset are inferred as registers.
- Seqshifts with both reset and set are inferred as registers.
- Seqshifts with enable signal having higher priority than synchronous set or synchronous reset are inferred as registers.

RAM Primitives Support in RTG4

The tool supports the following RAM primitives in the RTG4 device:

RAM1K18_RT

- Infer RAM1K18_RT for single-port, two-port, and dual-port synchronous read/write memory.
- Read-before-write in dual-port mode for single-port and dual-port synchronous memory.
- Read-enable extraction.

Limitation

Read-enable extraction for wide RAMs is not supported.

RAM64x18_RT

Infer RAM64x18_RT for single-port, two-port, and three-port synchronous/asynchronous read and synchronous write memory.

RAM Inference for SmartFusion2/IGLOO2/RTG4

RAM inference for SmartFusion2, IGLOO2, and RTG4 technologies is enhanced to use the BLK pin of the RAM for reducing power consumption. By setting the global option *low_power_ram_decomp 1* in the project file, the tool fractures the wide RAMs on the address width, using the BLK pin of the RAM to reduce power consumption. By default, the tool fractures wide RAMs by splitting the data width to improve timing.

This feature is supported on single-port, simple-dual port, and true-dual port RAM modes.

RAM Inference Enhancement for RTG4 (ECC Support)

RAM inference is enhanced for RTG4 using the Error Correction Codes (ECC) pin and the Single-Event Transient (SET) pin with error monitoring.

ECC is enabled through the `syn_ramstyle` attribute.

Attribute	Value	Description	Scope
<code>syn_ramstyle</code>	<code>ecc</code>	Enables RAM inference with ECC	FDC: global, view, instance HDL: module, architecture, memory

ECC Error Flag Generation

You can specify error flag monitoring in the FDC constraint file, at the instance level, using the tcl commands given below. The options in bold are mandatory.

Command	Argument	Description
syn_create_err_net	-name <new net name>	Specifies new net name to which the generated error flag is connected.
	-inst i:<RAM instance>	Specifies instance name of the high reliability module to be error monitored.
	[-single_bit -double_bit]	Specifies monitoring of single bit error flag or double bit error flag or both.
syn_connect	-from n:<new net name>	Specifies net name created through the <code>syn_create_err_net</code> command.
	-to {n:<existing net> t:<sub-module port> p:<top port>}	Specifies the destination for the generated error flag. It can be an existing net or a submodule output port or a top level output port.

Single bit and double bit error flag generation is controlled by the `-single_bit/-double_bit` arguments of the `syn_create_err_net` command.

-inst	-Single_bit -double_but	Description
i:<ECC RAM instance>	None Both	Create one error flag by ORing the following ports of SRAM blocks: A_SB_CORRECT B_SB_CORRECT A_DB_DETECT B_DB_DETECT
	- Single_bit	Create one error flag by ORing the following ports of SRAM blocks: A_SB_CORRECT B_SB_CORRECT
	- Double_bit	Create one error flag by ORing the following ports of SRAM blocks: A_DB_DETECT B_DB_DETECT

Example 1: With FDC Constraint

```

module test
  (clka, clkb, rst, wea, addra, dataaina, qa, web, addrb, datainb, qb, error_flag);
  parameter addr_width = 10;
  parameter data_width = 16;
  input  clka, clkb, wea, web, rst;
  input  [data_width - 1:0] dataaina, datainb;
  input  [addr_width - 1:0] addra, addrb;
  output error_flag;
  output reg [data_width - 1:0] qa, qb;
  reg [data_width - 1 : 0] mem [(2**addr_width) - 1:0]
  /* synthesis syn_ramstyle = "ecc" */;
  always @ (posedge clka)
  begin
    if (wea) mem[addra] <= dataaina;
  end
  always @ (posedge clkb)
  begin
    if (~rst)
      qa <= 16'd0;
    else begin

```



```

if(~wea)
  qa <= mem[addrb];
end
end
endmodule

```

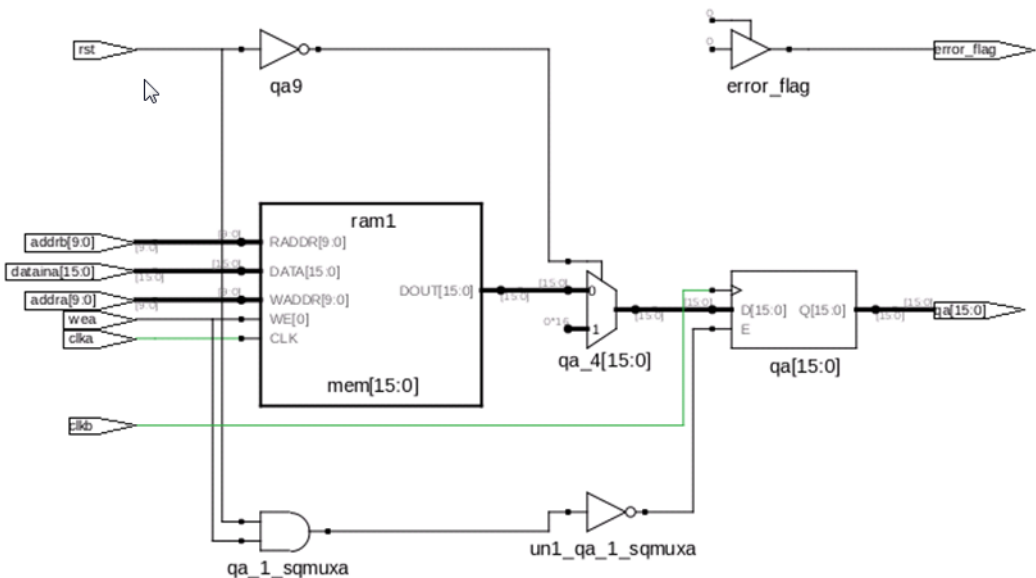
For the above RTL, specify one error flag for both single bit and double bit error, in the FDC file.

```

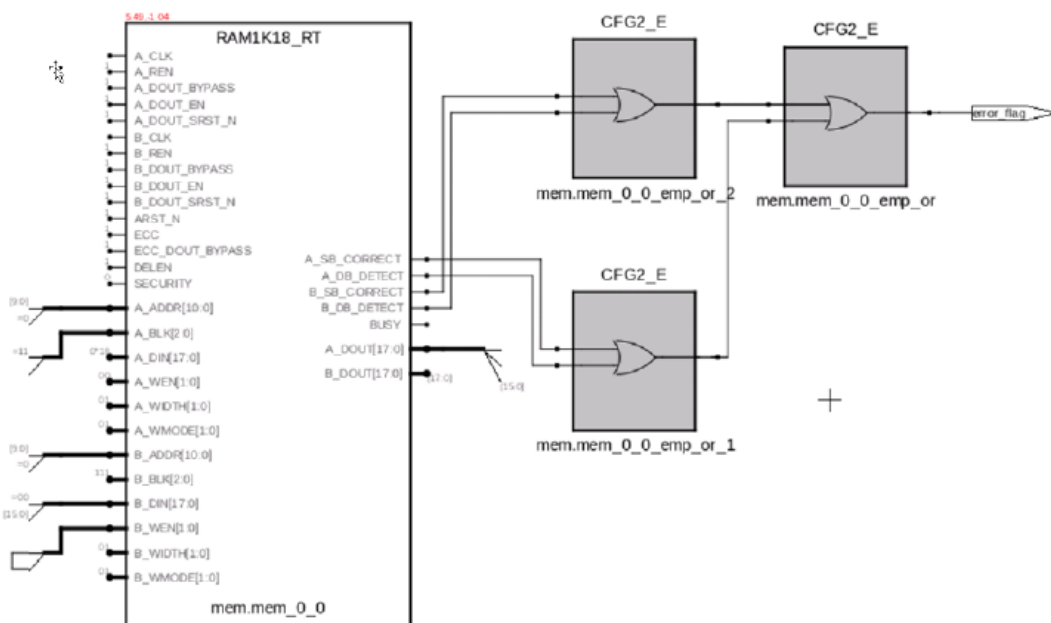
define_global_attribute {syn_ramstyle} {ecc}
syn_create_err_net {-name {error_net} -inst {i:mem[15:0]}}
syn_connect {-from {n:error_net} -to {p:error_flag} }

```

RTL View



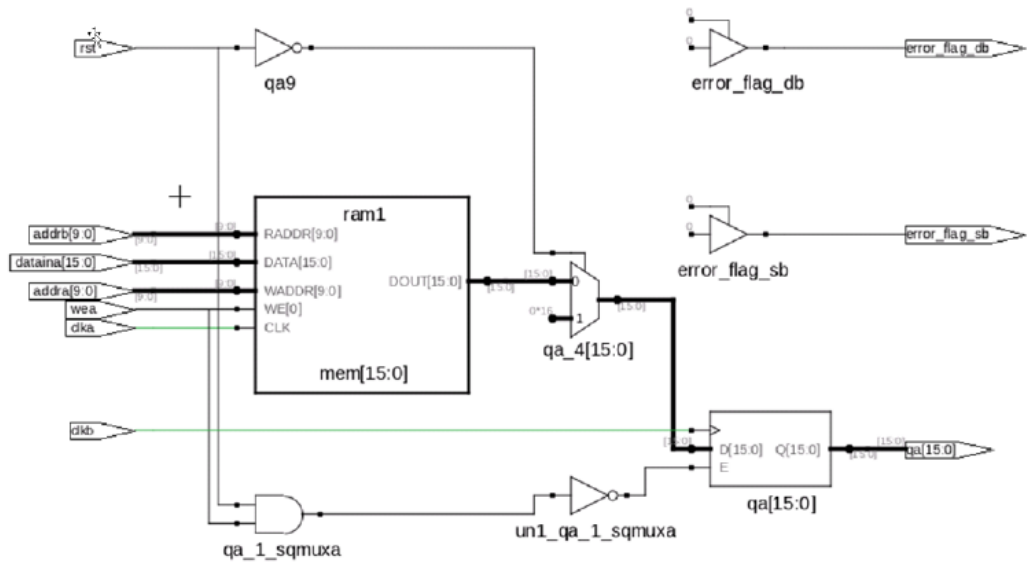
Technology View



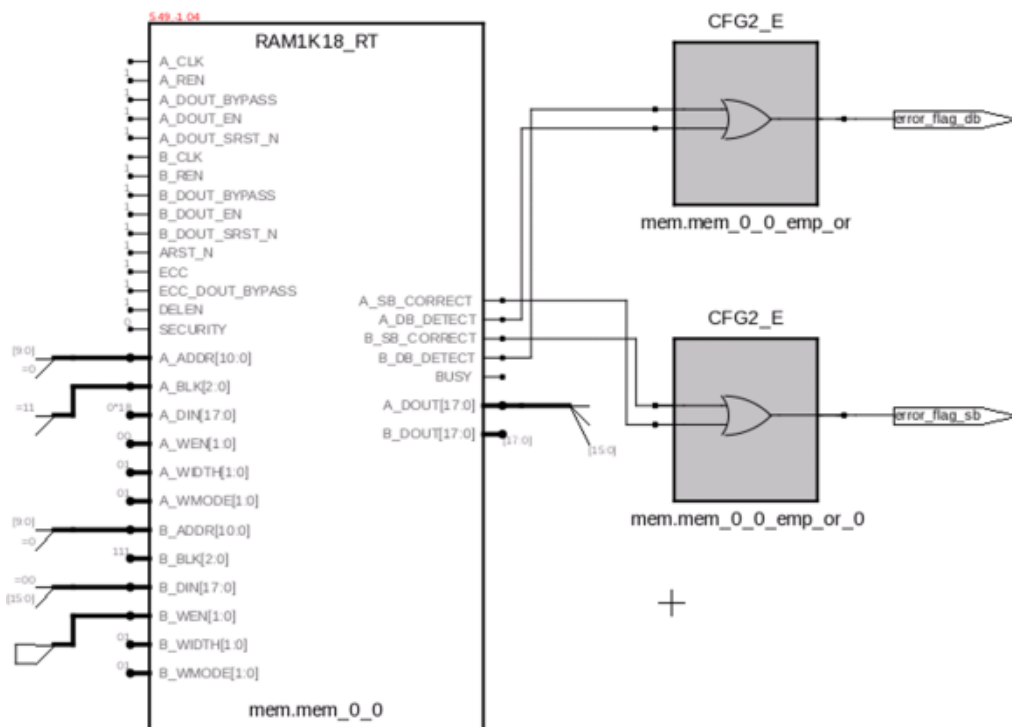
Example 2: With Single and Double Bit Error

For the RTL above, specify separate error flags for single bit and double bit errors.

```
define_attribute {i:mem[15:0]} {syn_ramstyle} {ecc}
syn_create_err_net {-name {error_net_sb} -inst {i:mem[15:0]} -
single_bit}
syn_connect {-from {n:error_net_sb} -to {p:error_flag_sb}}
syn_create_err_net {-name {db_error_net_db} -inst {i:mem[15:0]} -
double_bit}
syn_connect {-from {n:db_error_net_db} -to {p:db_error_flag_db} }
```

RTL View

Technology View



SET mitigation is also enabled using the `syn_ramstyle` attribute.

:

Attribute	Value	Description	Scope
<code>syn_ramstyle</code>	<code>ecc, set</code>	Enables RAM inference with ECC Enables SET mitigation	FDC: Global, view, instance HDL: module, architecture, memory

You can insert pipeline stages on the error flag path by including the options below in the `syn_create_error_net` command:

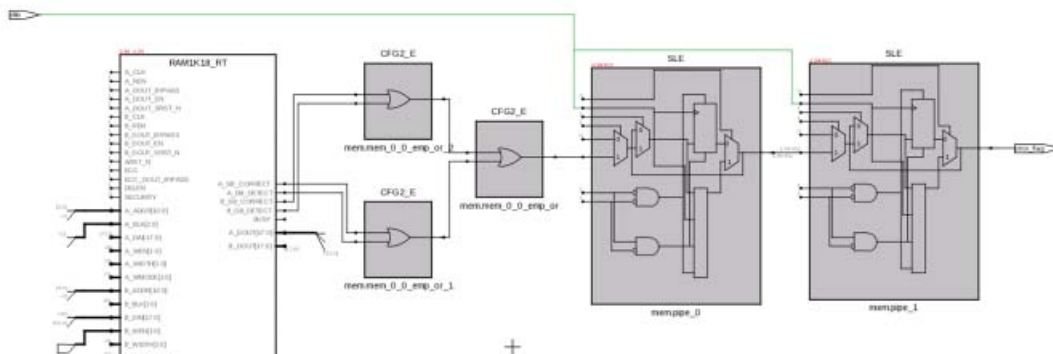
Command	Argument	Description
syn_create_err_net	-name <new net name>	Specifies new net name to which the generated error flag is connected.
	-inst i:<RAM instance>	Specifies instance name of the high reliability module to be error monitored.
	[-single_bit -double_bit]	Specifies monitoring of single-bit error flag or double-bit error flag or both.
	[-err_pipe_num {#}]	{#} Specifies number of pipeline stages to be introduced in the error flag path.
	[-err_clk {n:<clock to the pipeline registers>}]	Specifies clock signal for pipeline registers.

Example 3: With FDC Constraint

For the RTL above, insert two pipeline stages in the error flag path using the following FDC constraints:

```
define_global_attribute {syn_ramstyle} {ecc}
syn_create_err_net {-name {error_net} -inst {i:mem[15:0] -
err_pipe_num {2}
-err_clk {n:clkb}} }
syn_connect {-from {n:error_net} -to {p:error_flag} }
```

Technology View



Limitations

Inferring ECC pipeline stages (ECC_DOUT_BYPASS = 0) is not supported.

RAM Read Enable Extraction

RAM Read Enable extraction currently supports RAMs for output registers, with an enable. This feature is available for ProASIC3E devices only.

The following example contains a RAM with read enable.

```
`timescale 100 ps/100 ps
/* Synchronous write and read RAM */

module test (dout, addr, din, we, clk, ren);

    parameter data_width = 8;
    parameter address_width = 4;
    parameter ram_size = 16;

    output [data_width-1:0] dout;
    input [data_width-1:0] din;
    input [address_width-1:0] addr;
    input we, clk, ren;

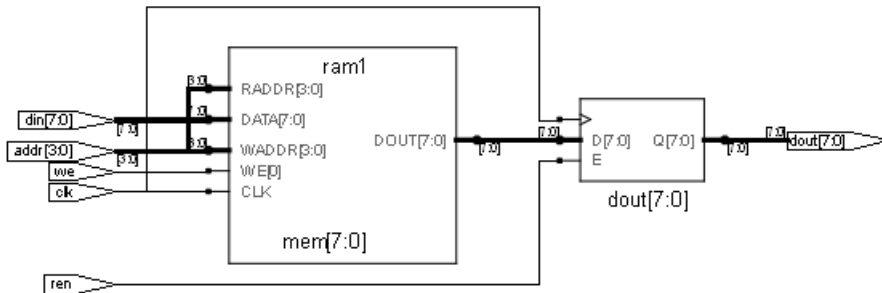
    reg [data_width-1:0] mem [ram_size-1:0];
    reg [data_width-1:0] dout;
```

```

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

endmodule

```



ProASIC3/3E/3L and IGLOO+/IGLOO/IGLOOe

The synthesis software extracts single-port and dual-port versions of the following RAM configurations:

RAM4K9	Synchronous write, synchronous read, transparent output
RAM512X18	Synchronous write, synchronous read, registered output

The architecture of the inferred RAM for the ProASIC3/3E/3L or IGLOO+/IGLOO/IGLOOe, can be registers, block_ram, rw_check, or no_rw_check. You set these values in the SCOPE interface using the syn_ramstyle attribute.

The following is an example of the RAM4K9 configuration:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is

```

```

port (q : out std_logic_vector(9 downto 0);
      d : in std_logic_vector(9 downto 0);
      addr : in std_logic_vector(9 downto 0);
      we : in std_logic;
      clk : in std_logic);
end ramtest;

architecture rtl of ramtest is
  type mem_type is array (1023 downto 0) of std_logic_vector
    (9 downto 0);

  signal mem : mem_type;
  signal read_addr : std_logic_vector(9 downto 0);

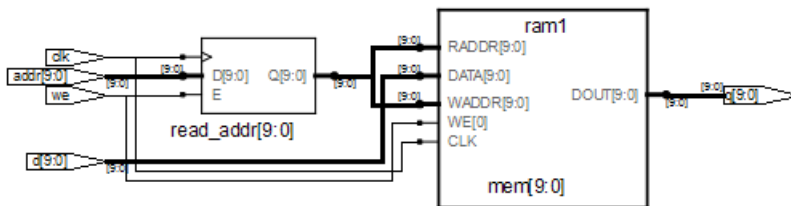
begin

  q <= mem(conv_integer(read_addr));

  process (clk) begin
    if rising_edge(clk) then
      if (we = '1') then
        read_addr <= addr;
        mem(conv_integer(read_addr)) <= d;
      end if;
    end if;
  end process;

end rtl;

```



The following is an example of the RAM512X18 configuration:

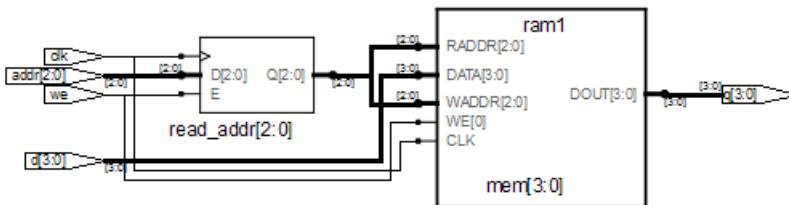
```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
port (q : out std_logic_vector(3 downto 0);
      d : in std_logic_vector(3 downto 0);
      addr : in std_logic_vector(2 downto 0);
      we : in std_logic;
      clk : in std_logic);
end ramtest;

architecture rtl of ramtest is
type mem_type is array (7 downto 0) of
  std_logic_vector (3 downto 0);
signal mem : mem_type;
signal read_addr : std_logic_vector(2 downto 0);
begin
q <= mem(conv_integer(read_addr));
process (clk) begin
  if rising_edge(clk) then
    if (we = '1') then
      read_addr <= addr;
      mem(conv_integer(read_addr)) <= d;
    end if;
  end if;
end process;
end rtl;

```



SmartFusion2

SmartFusion2 devices support two types of RAM macros: RAM1K18 and RAM64X18. The synthesis software extracts the RAM structure from the RTL and infers RAM1K18 or RAM64X18 based on the size of the RAM.

The default criteria for specifying the macro is described in the table below for the following RAM types.

True Dual-Port Synchronous Read Memory	The synthesis tool maps to RAM1K18, regardless of its memory size.
Simple Dual-Port or Single-Port Synchronous Memory	<p>If the size of the memory is:</p> <ul style="list-style-type: none"> • 4608 bits or greater, the synthesis tool maps to RAM1K18. • Greater than 12 bits and less than 4608 bits, the synthesis tool maps to RAM64X18. • Less than or equal to 12 bits, the synthesis tool maps to registers.
Simple Dual-Port or Single-Port Asynchronous Memory	When the size of the memory is 12 bits or greater, the synthesis tool maps to RAM64x18. Otherwise, it maps to registers.
Three Port RAM Inference Support	<p>This feature is supported on SmartFusion2 and IGLOO2 devices only.</p> <p>RAM64x18 is a 3-port memory providing one Write port and two Read ports.</p> <p>Write operation is synchronous while read operations can be asynchronous or synchronous.</p> <p>The tool infers RAM64X18 for such structures.</p>

You can override the default behavior by applying the `syn_ramstyle` attribute to control how the memory gets mapped. To map to

- RAM1K18 set `syn_ramstyle = "lsram"`
- RAM64X18 set `syn_ramstyle = "uram"`
- Registers set `syn_ramstyle = "registers"`

The value you set for this attribute always overrides the default behavior.

Three Port RAM inference support

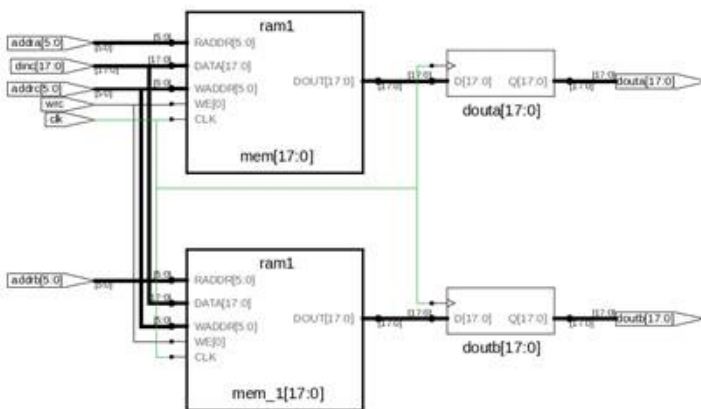
Example 1) Three Port RAM Verilog Example—Synchronous Read

```
module
ram_infer15_rtl(clk,dinc,douta,doutb,wrc,rda,rdb,addra,addrb,addrc
);
input clk;
input [17:0] dinc;
input wrc,rda,rdb;
input [5:0] addra,addrb,addrc;
output [17:0] douta,doutb;
reg [17:0] douta,doutb;
reg [17:0] mem [0:63];
always@(posedge clk)
begin
if(wrc)
mem[addrc] <= dinc;
end

always@(posedge clk)
begin
douta <= mem[addra];
end

always@(posedge clk)
begin
doutb <= mem[addrb] ;
end
endmodule
```

RTL view:



The tool infers one RAM64X18.

Example 2) Three Port RAM VHDL Example—Asynchronous Read

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_singleport_noreg is
port (d : in std_logic_vector(7 downto 0);
      addw : in std_logic_vector(6 downto 0);
      addr1 : in std_logic_vector(6 downto 0);
      addr2 : in std_logic_vector(6 downto 0);
      we : in std_logic;
      clk : in std_logic;
      q1 : out std_logic_vector(7 downto 0);
      q2 : out std_logic_vector(7 downto 0) );
end ram_singleport_noreg;
architecture rtl of ram_singleport_noreg is
type mem_type is array (127 downto 0) of
std_logic_vector (7 downto 0);
signal mem: mem_type;
begin
process (clk)
begin
if rising_edge(clk) then
if (we = '1') then
mem(conv_integer (addw)) <= d;
end if;

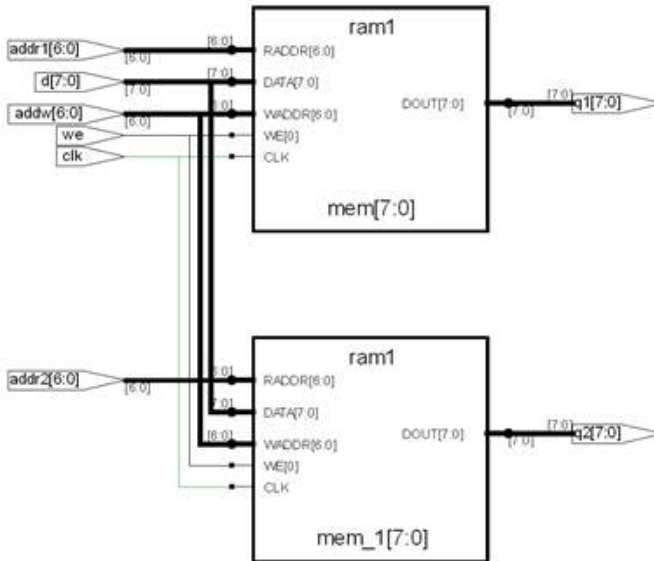
```

```

end if;
end process;
q1<= mem(conv_integer (addr1));
q2<= mem(conv_integer (addr2));
end rtl;

```

RTL View:



The tool infers one RAM64X18.

Instantiating RAMs with SYNCORE

The SYNCORE Memory Compiler is available under the IP Wizard to help you generate HDL code for your specific RAM implementation requirements. For information on using the SYNCORE Memory Compiler, see [Specifying RAMs with SYNCORE](#), on page 456 in the *User Guide*.

Control Signals Extraction for Registers (SLE)

The synthesis software supports extraction of control signals, enable, synchronous set or reset and asynchronous reset on the registers. The tool packs enable using EN pin, synchronous set or reset using SLn pin, and asynchronous reset using ALn pin of SLE.

When the fanout limit is 12, synchronous set or reset is packed using SLn pin. When the fanout limit is less than 12, the tool inserts extra logic for synchronous set or reset.

The tool supports the packing of enable signal with higher priority than the reset signal (synchronous) into SLE.

Output Files and Forward-annotation for Microsemi

After synthesis, the software generates a log file and output files for Microsemi. The following describe some of the reports with Microsemi-specific information, or files that forward annotate information for the Microsemi P&R tools.

- [VM Flow Support, on page 407](#)
- [Forward-annotating Constraints for Placement and Routing, on page 408](#)
- [Synthesis Reports, on page 409](#)

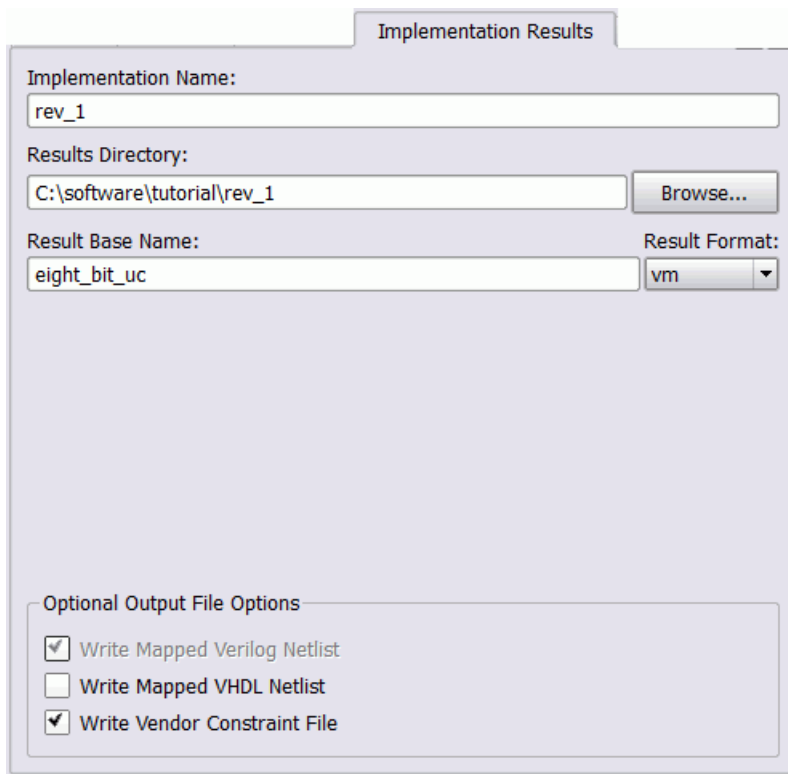
VM Flow Support

The tool generates a Verilog output netlist (*.vm*) for SmartFusion2 and IGLOO2 for P&R flow. After synthesis, the tool:

- Writes a separate SDC file (**_vm.sdc*).
- Write a separate TCL file (**_partition_vm.tcl*) to forward-annotate the timestamps on instances in incremental compile point flow.
- Forward-annotates properties like RTL attributes in *.vm* netlist and constraints in SDC file.

By default, the tool generates a *.edn* netlist. You can change the netlist from EDIF to Verilog.

To select Verilog output netlist, go to Implementation Options->Implementation Results->Result Format. Select *vm* from the drop-down menu, click OK, and save the project.



The image shows a software dialog box titled "Implementation Results". It contains several input fields and a section for optional output file options. The "Implementation Name" field is set to "rev_1". The "Results Directory" field is set to "C:\software\tutorial\rev_1" with a "Browse..." button next to it. The "Result Base Name" field is set to "eight_bit_uc" and the "Result Format" dropdown menu is set to "vm". At the bottom, there is a section titled "Optional Output File Options" with three checkboxes: "Write Mapped Verilog Netlist" (checked), "Write Mapped VHDL Netlist" (unchecked), and "Write Vendor Constraint File" (checked).

Implementation Results	
Implementation Name: rev_1	
Results Directory: C:\software\tutorial\rev_1	Browse...
Result Base Name: eight_bit_uc	Result Format: vm
Optional Output File Options	
<input checked="" type="checkbox"/> Write Mapped Verilog Netlist	
<input type="checkbox"/> Write Mapped VHDL Netlist	
<input checked="" type="checkbox"/> Write Vendor Constraint File	

Forward-annotating Constraints for Placement and Routing

For Microsemi Fusion, IGLOO, ProASIC3, and SmartFusion technology families, the synthesis tool forward annotates timing constraints to placement and routing through an Microsemi constraint (*filename_sdc.sdc*) file. These timing constraints include clock period, max delay, multiple-cycle paths, input and output delay, and false paths. During synthesis, the Microsemi constraint file is generated using synthesis tool attributes and constraints.

By default, Microsemi constraint files are generated. You can disable this feature in the Project view. To do this, bring up the Implementation Options dialog box (Project -> Implementation Options), then, on the Implementation Results panel, disable Write Vendor Constraint File.

Forward-annotated Constraints

The constraint file generated for Microsemi's place-and-route tools has an `_sdc.sdc` file extension. Constraints files that properly specify either Synplify-style timing constraints or Synopsys SDC timing constraints can be forward annotated to support the Microsemi P&R tool.

Synthesis	Microsemi
create_clock	<p><code>create_clock</code></p> <p>The <code>create_clock</code> constraint is allowed for all NGT families. No wildcards are accepted. The pin or port must exist in the design.</p> <p>The <code>-name</code> argument is not supported as that would define a virtual clock. However, for backward compatibility, a <code>-name</code> argument does not generate an error or warning when encountered in an <code>.sdc</code> file.</p>
set_max_delay	<p><code>set_max_delay</code></p> <p>The <code>set_max_delay</code> constraint is allowed for all NGT families. Wildcards are accepted.</p>
set_multicycle_path	<p><code>set_multicycle_path</code></p> <p>You must specify at least one of the <code>-from</code> or <code>-to</code> arguments, however, it is best to specify both. Wildcards are accepted.</p> <p>Multicycle constraints with <code>-from</code> and/or <code>-to</code> arguments only are supported for Microsemi ProASIC3/3E technologies. Multicycle constraints with a <code>-through</code> argument are not supported for any NGT family.</p>
set_false_path	<p><code>set_false_path</code></p> <p>Only false path constraints with a <code>-through</code> argument are supported for NGT families. False path constraints with either <code>-from</code> and/or <code>-to</code> arguments are not supported for any NGT family. Wildcards are accepted.</p>

Synthesis Reports

The synthesis tool generates a resource usage report, a timing report, and a net buffering report for the Microsemi designs that you synthesize. To view the synthesis reports, click View Log.

Optimizations for Microsemi Designs

The synthesis tools offer various optimizations for Microsemi designs. The following describe the optimizations in more detail:

- [The syn_maxfan Attribute in Microsemi Designs, on page 410](#)
- [Promote Global Buffer Threshold, on page 411](#)
- [I/O Insertion, on page 412](#)
- [Number of Critical Paths, on page 413](#)
- [Retiming, on page 413](#)
- [Update Compile Point Timing Data Option, on page 413](#)
- [Operating Condition Device Option, on page 415](#)
- [Radiation-tolerant Applications, on page 417](#)

The syn_maxfan Attribute in Microsemi Designs

The syn_maxfan attribute is used to control the maximum fanout of the design, or an instance, net, or port. The limit specified by this attribute is treated as a hard or soft limit depending on where it is specified. The following rules described the behavior:

- Global fanout limits are usually specified with the fanout guide options (Project->Implementation Options->Device), but you can also use the syn_maxfan attribute on a top-level module or view to set a global soft limit. This limit may not be honored if the limit degrades performance. To set a global hard limit, you must use the Hard Limit to Fanout option.
- A syn_maxfan attribute can be applied locally to a module or view. In this case, the limit specified is treated as a soft limit for the scope of the module. This limit overrides any global fanout limits for the scope of the module.
- When a syn_maxfan attribute is specified on an instance that is not of primitive type inferred by Synopsys FPGA compiler, the limit is considered a soft limit which is propagated down the hierarchy. This attribute overrides any global fanout limits.

- When a `syn_maxfan` attribute is specified on a port, net, or register (or any primitive instance), the limit is considered a hard limit. This attribute overrides any other global fanout limits. Note that the `syn_maxfan` attribute does not prevent the instance from being optimized away and that design rule violations resulting from buffering or replication are the responsibility of the user.

Promote Global Buffer Threshold

The Promote Global Buffer Threshold option is for the SmartFusion, Fusion, IGLOO, and ProASIC3 technology families only. This option is used for both ports and nets.

The Tcl command equivalent is `set_option -globalthreshold value`, where the value refers to the minimum number of fanout loads. The default value is 1.

Only signals with fanout loads larger than the defined value are promoted to global signals. The synthesis tool assigns the available global buffers to drive these signals using the following priority:

1. Clock
2. Asynchronous set/reset signal
3. Enable, data

SmartFusion2, IGLOO2, and RTG4 Global Buffer Promotion

The synthesis software inserts the global buffer (CLKINT) on clock, asynchronous set/reset, and data nets based on a threshold value. SmartFusion2, IGLOO2, and RTG4 devices have specific threshold values that cannot be changed for the different types of nets in the design. Inserting global buffers on nets with fanout greater than the threshold can help reduce the route delay during place and route.

The threshold values for SmartFusion2 and IGLOO2 devices are the following:

Net	Global buffer inserted for threshold value > or =
Clock	2
Asynchronous Set/Reset	12
Data	5000

The threshold values for RTG4 devices are the following:

Net	Global buffer inserted for threshold value > or =
Clock	2
Asynchronous Set/Reset	200000
Data	5000

To override these default option settings you can:

- Use the `syn_noclockbuf` attribute on a net that you do not want a global buffer inserted, even though fanout is greater than the threshold.
- Use `syn_insert_buffer="CLKINT"` so that the tool inserts a global buffer on the particular net, which is less than the threshold value. You can specify `CLKINT`, `RCLKINT`, `CLKBUF`, or `CLKBIBUF` as values for SmartFusion2, RTG4, and IGLOO2 devices.

I/O Insertion

The Synopsys FPGA synthesis tool inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist unless you disable I/O insertion. You can control I/O insertion with the Disable I/O Insertion option (Project->Implementation Options->Device).

If you do not want to automatically insert any I/O pads, check the Disable I/O Insertion box (Project->Implementation Options->Device). This is useful to see how much area your blocks of logic take up, before synthesizing an entire FPGA. If you disable automatic I/O insertion, you will not get any I/O pads in your design unless you manually instantiate them yourself.

If you disable I/O insertion, you can instantiate the Microsemi I/O pads you need directly. If you manually insert I/O pads, you only insert them for the pins that require them.

Number of Critical Paths

The Max number of critical paths in SDF option (Project->Implementation Options->Device) is only available for the SmartFusion, Fusion, IGLOO, and ProASIC3 technology families. It lets you set the maximum number of critical paths in a forward-annotated constraint (sdf) file. The sdf file displays a prioritized list of the worst-case paths in a design. Microsemi Designer prioritizes routing to ensure that the worst-case paths are routed efficiently.

The default value for the number of critical paths that are forward annotated is 4000. Various design characteristics affect this number, so experiment with a range of values to achieve the best circuit performance possible.

Retiming

Retiming is the process of automatically moving registers (register balancing) across combinational gates to improve timing, while ensuring identical logic behavior. Currently retiming is available for the SmartFusion, Fusion, IGLOO, and ProASIC3 technology families.

You enable/disable global retiming with the Retiming device mapping option (Project view or Device panel). You can use the `syn_allow_retiming` attribute to enable or disable retiming for individual flip-flops. See [syn_allow_retiming, on page 46](#) and the *User Guide* for more information.

Update Compile Point Timing Data Option

In SmartFusion, Fusion, IGLOO, and ProASIC3 design families, the Synopsys FPGA compile-point synthesis flow lets you break down a design into smaller synthesis units, called *compile points*, making incremental synthesis possible. See [Synthesizing Compile Points, on page 433](#) in the *User Guide*.

The Update Compile Point Timing Data option controls whether or not changes to a locked compile point force remapping of its parents, taking into account the new timing model of the child.

Note: To simplify this description, the term *child* is used here to refer to a compile point that is contained inside another; the term *parent* is used to refer to the compile point that contains the child. These terms are thus not used here in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points.

Disabled

When the Update Compile Point Timing Data option is *disabled* (the default), only (locked) compile points that have changed are remapped, and their remapping does *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used, instead, to map and optimize its parents.

An exceptional case occurs when the option is disabled and the *interface* of a locked compile point is changed. Such a change requires that the immediate parent of the compile point be changed accordingly, so both are remapped. In this exceptional case, however, the *updated* timing model (not the old model) of the child is used when remapping this parent.

Enabled

When the Update Compile Point Timing Data option is *enabled*, locked compile-point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account (because the option was disabled).

The timing model of a compile point is updated when either of the following is true:

- The compile point is remapped, and the Update Compile Point Timing Data option is enabled.
- The interface of the compile point is changed.

Operating Condition Device Option

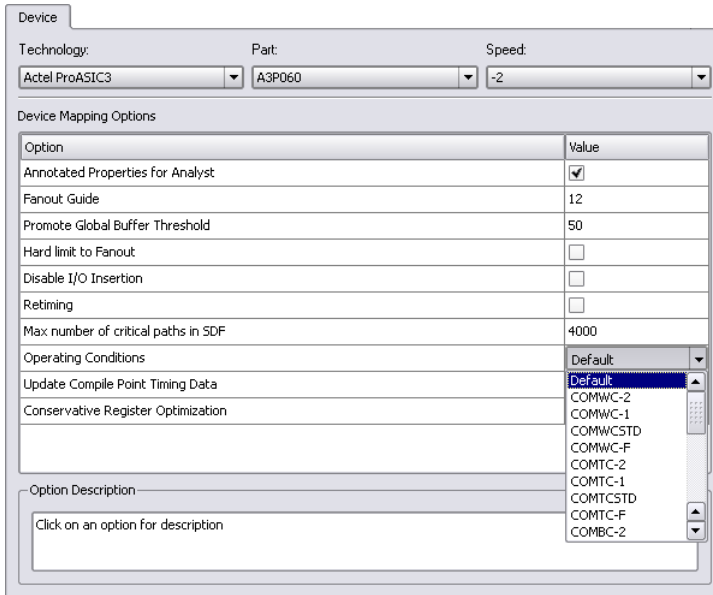
You can specify an operating condition for certain Microsemi technologies:

- ProASIC3/3E/3L
- IIGLOO2/IGLOO+/IGLOO/IGLOOe
- SmartFusion2, SmartFusion, FUSION
- RTG4

Different operating conditions cause differences in device performance. The operating condition affects the following:

- optimization, if you have timing constraints
- timing analysis
- timing reports

To set an operating condition, select the value for Operating Conditions from the menu on the Device tab of the Implementation Options dialog box.



To set an operating condition in a project or Tcl file, use the command:

```
set_option -opcond value
```

where *value* can be specified like the following typical operating conditions:

Default	Typical timing
MIL-WC	Worst-case Military timing
MIL-TC	Typical-case Military timing
MIL-BC	Best-case Military timing
Automotive-WC	Worst-case Automotive timing

For Example

The Microsemi operating condition can contain any of the following specifications:

- MIL—military
- COM—commercial

- IND—Industrial
- TGrade1
- TGrade2

as well as, include one of the following designations:

- WC—worst case
- BC—best case
- TC—typical case

For specific operating condition values for your required technology, see the Device tab on the Implementation Options dialog box.

Even when a particular operating condition is valid for a family, it may not be applicable to every part/package/speed-grade combination in that family. Consult Microsemi's documentation or software for information on valid combinations and more information on the meaning of each operating condition.

Radiation-tolerant Applications

You can specify the radiation-resistant design technique to use on an object for a design with the `syn_radhardlevel` attribute. This attribute can be applied to a module/architecture or a register output signal (inferred register in VHDL), and is used in conjunction with the Microsemi macro files supplied with the software.

Values for `syn_radhardlevel` are as follows:

Value	Description
none	Standard design techniques are used.
cc	Combinational cells with feedback are used to implement storage rather than flip-flop or latch primitives.
tmr	Triple module redundancy or triple voting is used to implement registers. Each register is implemented by three flip-flops or latches that “vote” to determine the state of the register.
tmr_cc	Triple module redundancy is used where each voting register is composed of combinational cells with feedback rather than flip-flop or latch primitives

For details, see:

- [Working with Radhard Designs, on page 541](#)
- [syn_radhardlevel, on page 170](#)

Integration with Microsemi Tools and Flows

The following describe how the synthesis tools support various tools and flows for Microsemi designs:

- [Compile Point Synthesis, on page 419](#)
- [Incremental Synthesis Flow, on page 420](#)
- [Microsemi Place-and-Route Tools, on page 420](#)

Compile Point Synthesis

Compile-point synthesis is available only for the Microsemi SmartFusion, SmartFusion2, RTG4, Fusion, IGLOO+/IGLOO/IGLOOe/IGLOO2, and ProASIC3/3E/3L technology families. The compile-point synthesis flow lets you achieve incremental design and synthesis without having to write and maintain sets of complex, error-prone scripts to direct synthesis and keep track of design dependencies. See [Synthesizing Compile Points, on page 433](#) for a description, and [Working with Compile Points, on page 415](#) in the *User Guide* for a step-by-step explanation of the compile-point synthesis flow.

In device technologies that can take advantage of compile points, you break down your design into smaller synthesis units or *compile points*, in order to make incremental synthesis possible. A compile point is a module that is treated as a block for incremental mapping: When your design is resynthesized, compile points that have already been synthesized are not resynthesized, unless you have changed:

- the HDL source code in such a way that the design logic is changed,
- the constraints applied to the compile points, or
- the device mapping options used in the design.

(For details on the conditions that necessitate resynthesis of a compile point, see [Compile Point Basics, on page 416](#), and [Update Compile Point Timing Data Option, on page 413](#).)

Incremental Synthesis Flow

Microsemi IGLOO2 , SmartFusion2, and RTG4 Technologies

The synthesis tool provides timestamps for each manual compile point in the *_partition.tcl file. You can use the timestamps to check whether the compile point was resynthesized in an incremental run of the tool.

To run this flow:

1. Define compile point constraint on the modules in the design. For example:

```
define_compile_point {viewName} -type {locked, partition}  
-cpfile {fileName}
```

2. Run the standard synthesis flow. The synthesis tool writes the timestamps for each compile point in the *designName_partition.tcl* file. For example:

```
set_partition_info -name partitionName -timestamp timestamp
```

For an incremental synthesis run, only affected compile points display new timestamps, while unaffected compile points retain the same timestamps.

Check the Compile Point Summary report available in the log file.

Microsemi Place-and-Route Tools

You can run place and route automatically after synthesis. For details on how to set options, see [Running P&R Automatically after Synthesis, on page 550](#) in the *User Guide*.

For details about the place-and-route tools, refer to the Microsemi documentation.

Microsemi Device Mapping Options

You select device mapping options for Microsemi technologies, select Project -> Implementation Options->Device and set the options.

Option	For details, see ...
Conservative Register Optimization	See the Microsemi Tcl set_option Command Options , on page 423 for more information about the <code>preserve_registers</code> option.
Disable I/O Insertion	I/O Insertion , on page 412.
Fanout Guide	Setting Fanout Limits , on page 398 of the <i>User Guide</i> and The syn_maxfan Attribute in Microsemi Designs , on page 410.
Hard Limit to Fanout	Setting Fanout Limits , on page 398 of the <i>User Guide</i> and The syn_maxfan Attribute in Microsemi Designs , on page 410.
Max number of critical paths in SDF (certain technologies)	Number of Critical Paths , on page 413.
Operating Conditions (certain technologies)	Operating Condition Device Option , on page 415
Promote Global Buffer Threshold	Controlling Buffering and Replication , on page 400 of the <i>User Guide</i> and Promote Global Buffer Threshold , on page 411.

Option	For details, see ...
Read Write Check on RAM	<p>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.</p> <p>For details about using this option in conjunction with the <code>syn_ramstyle</code> attribute, see syn_ramstyle, on page 173.</p>
Retiming	<p>Retiming, on page 386 of the <i>User Guide</i> and Retiming, on page 413.</p>
Resolve Mixed Drivers	<p>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.</p>
Update Compile point Timing Data	<p>Update Compile Point Timing Data Option, on page 413</p>

Microsemi Tcl set_option Command Options

You can use the `set_option` Tcl command to specify the same device mapping options as are available through the Implementation Options dialog box displayed in the Project view with Project -> Implementation Options (see [Implementation Options Command](#), on page 204).

This section describes the Microsemi-specific `set_option` Tcl command options. These include the target technology, device architecture, and synthesis styles.

The table below provides information on specific options for Microsemi architectures. For a complete list of options for this command, refer to [set_option](#), on page 55. You cannot specify a package (`-package` option) for some Microsemi technologies in the synthesis tool environment. You must use the Microsemi back-end tool for this.

Option	Description
-technology <i>keyword</i>	Sets the target technology for the implementation. Keyword must be one of the following Microsemi architecture names: FUSION, IGLOO, IGLOOE, IGLOO+, ProASIC3, ProASIC3E, ProASIC3L, SmartFusion, and SmartFusion2.
-part <i>partName</i>	Specifies a part for the implementation. Refer to the Implementation Options dialog box for available choices.
-package <i>packageName</i>	Specifies the package. Refer to Project-> Implementation Options->Device for available choices.
-speed_grade <i>value</i>	Sets the speed grade for the implementation. Refer to the Implementation Options dialog box for available choices.
-disable_io_insertion 1 0	Prevents (1) or allows (0) insertion of I/O pads during synthesis. The default value is <code>false</code> (enable I/O pad insertion). For additional information about disabling I/O pads, see I/O Insertion , on page 412.
-fanout_limit <i>value</i>	Sets the fanout limit guideline for the current project. For more information about fanout limits, see The syn_maxfan Attribute in Microsemi Designs , on page 410.

Option	Description
-globalthreshold <i>value</i>	Sets the minimum fanout load value. This option applies only to the SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, and ProASIC3/3E/3L technologies. For more information, see Promote Global Buffer Threshold , on page 411.
-maxfan_hard 1	Specifies that the specified -fanout_guide value is a hard fanout limit that the synthesis tool must not exceed. To set a guideline limit, see the -fanout_guide option. For more information about fanout limits, see The syn_maxfan Attribute in Microsemi Designs , on page 410.
-opcond <i>value</i>	Sets the operating condition for device performance in the areas of optimization, timing analysis, and timing reports. This option applies only to the SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, and ProASIC3/3E/3L technologies. Values are Default, MIL-WC, IND-WC, COM-WC, and Automotive-WC. See Operating Condition Device Option , on page 415 for more information.
-preserve_registers 1 0	When enabled, the software uses less restrictive register optimizations during synthesis if area is not as great a concern for your device. The default for this option is disabled (0).
-resolve_multiple_driver 1 0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
-report_path <i>value</i>	Sets the maximum number of critical paths in a forward-annotated SDF constraint file. This option applies only to the SmartFusion, Fusion, IGLOO2, IGLOO+/IGLOO/IGLOOe, and ProASIC3/3E/3L technologies. For information about setting critical paths, see Number of Critical Paths , on page 413.
-retiming 1 0	<i>SmartFusion, Fusion, IGLOO2, GLOO+/IGLOO/IGLOOe, and ProASIC3/3E/3L</i> When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). For additional information about retiming, see Retiming , on page 413

Option	Description
-RWCheckOnRam 1 0	<p>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.</p> <p>For more information about using this option in conjunction with the syn_ramstyle attribute, see syn_ramstyle, on page 173.</p>
-update_models_cp 1 0	<p><i>SmartFusion, Fusion, IGLOO2, IGLOO+/IGLOO/IGLOOe, and ProASIC3/3E/3L</i></p> <p>Determines whether (1) or not (0) changes to a locked compile point force remapping of its parents, taking into account the new timing model of the child. See Update Compile Point Timing Data Option, on page 413, for details.</p>

Microsemi Attribute and Directive Summary

The following table summarizes the synthesis and Microsemi-specific attributes and directives available with the Microsemi technology. Complete descriptions and examples are in [Attributes and Directives Summary, on page 13](#).

Attribute/Directive	Description
alsloc	Forward annotates the relative placements of macros and IP blocks to Microsemi Designer.
alspin	Assigns scalar or bus ports to Microsemi I/O pin numbers.
alspreserve	Specifies that a net be preserved, and prevents it from being removed during place-and-route optimization.
black_box_pad_pin (D)	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins (D)	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
full_case (D)	Specifies that a Verilog case statement has covered all possible cases.
loop_limit (D)	Specifies a loop iteration limit for for loops.
parallel_case (D)	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
syn_allow_retiming	Specifies whether registers can be moved during retiming.
syn_black_box (D)	Defines a black box for synthesis.
syn_encoding	Specifies the encoding style for state machines.
syn_enum_encoding (D)	Specifies the encoding style for enumerated types (VHDL only).

(D) indicates directives; all others are attributes.

Attribute/Directive	Description
syn_global_buffers	Sets the number of global buffers to use in a ProASIC3/3E/3L.
syn_hier	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
syn_insert_buffer	Inserts a clock buffer according to the specified value.
syn_insert_pad	Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.
syn_isclock (D)	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_keep (D)	Prevents the internal signal from being removed during synthesis and optimization.
syn_maxfan	Overrides the default fanout guide for an individual input port, net, or register output.
syn_multstyle	Determines how multipliers are implemented for Microsemi devices.
syn_netlist_hierarchy	Determines whether the EDIF output netlist is flat or hierarchical.
syn_noarrayports	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.
syn_noclockbuf	Turns off the automatic insertion of clock buffers.
syn_noprune (D)	Controls the automatic removal of instances that have outputs that are not driven.
syn_pad_type	Specifies an I/O buffer standard.
syn_preserve (D)	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
syn_probe	Adds probe points for testing and debugging.
syn_radhardlevel	Specifies the radiation-resistant design technique to apply to a module, architecture, or register.

(D) indicates directives; all others are attributes.

Attribute/Directive	Description
syn_ramstyle	Specifies the implementation to use for an inferred RAM. You apply <code>syn_ramstyle</code> globally, to a module, or to a RAM instance.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
syn_replicate	Controls replication.
syn_resources	Specifies resources used in black boxes.
syn_sharing (D)	Specifies resource sharing of operators.
syn_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_state_machine (D)	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n> (D)	Defines timing clock to output delay through the black box. The <i>n</i> indicates a value between 1 and 10.
syn_tpd<n> (D)	Specifies timing propagation for combinational delay through the black box. The <i>n</i> indicates a value between 1 and 10.
syn_tristate (D)	Specifies that a black-box pin is a tristate pin.
syn_tsu<n> (D)	Specifies the timing setup delay for input pins, relative to the clock. The <i>n</i> indicates a value between 1 and 10.
translate_off/translate_on (D)	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

(D) indicates directives; all others are attributes.

APPENDIX B

RAM Example Code

This appendix contains the code samples that are referenced by the corresponding chapter.

Example 1A: 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
```

[Back](#)

Example 1B: 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
```

[Back](#)

Index

Symbols

- _ta.srm file [252](#)
- .adc file [242](#)
- .areasrr file [249](#)
- .edf file [250](#)
- .fse file [249](#)
- .info file [249](#)
- .ini file [242](#)
- .prj file [242](#)
- .sap
 - annotated properties for analyst [251](#)
- .sar file [251](#)
- .sdc file [242](#)
- .srd file [251](#)
- .srm file [251](#)
- .srr file [254](#)
 - watching selected information [44](#)
- .srs file [252](#)
 - initial values (Verilog) [297](#)
- .sv file [243](#)
 - SystemVerilog source file [243](#)
- .ta file
 - See [timing report file 252](#)
- .v file [243](#)
- .vhd file [243](#)
- .vhm file [254](#)
- .vm file [254](#)

A

- adc file (analysis design constraint) [242](#)
- adder
 - SYNCore [342](#)
- Allow Docking command [45](#)
- Alt key, selecting columns in Text Editor [62](#)
- analysis design constraint file (.adc) [242](#)
- Analyst toolbar [78](#)
- annotated properties for analyst
 - .sap [251](#)

- .timing annotated properties (.tap) [253](#)
- archive file (.sar) [251](#)
- areasrr file
 - hierarchical area report [267](#)
- arrow keys, selecting objects in Hierarchy
 - Browser [119](#)
- arrow pointers for push and pop [117](#)
- asynchronous clock report
 - description [266](#)
- attributes (Microsemi) [426](#)
- Attributes demo [64](#)
- Attributes panel, SCOPE [170](#)
- auto constraints [149](#)
 - Maximize option [93](#)

B

- batch mode [23](#)
- black boxes
 - See also [macros, macro libraries](#)
 - Microsemi [381](#)
- block RAM
 - dual-port RAM examples [286](#)
 - NO_CHANGE mode example [282](#)
 - READ_FIRST mode example [281](#)
 - single-port RAM examples [283](#)
 - WRITE_FIRST mode example [279](#)
- block RAMs
 - syn_ramstyle attribute [428](#)
- bus_dimension_separator_style command
 - [236](#)
- bus_naming_style command [236](#)
- buttons and options, Project view [92](#)

C

- c_diff command (collections) [163](#)
- c_intersect command (collections) [163](#)
- c_print command (collections) [162](#)
- c_sub command (collections) [163](#)

- c_symdiff command (collections) [163](#)
 - c_union command (collections) [162](#)
 - callback functions, customizing flow [370](#)
 - cck.rpt file (constraint checking report) [249](#)
 - check boxes, Project view [92](#)
 - clock buffering report, log file (.srr) [256](#)
 - clock frequency goals, tradeoffs using different [372](#)
 - clock groups
 - Clock Relationships (timing report) [262](#)
 - clock groups, SCOPE [155](#)
 - clock paths, ignoring [185](#)
 - clock pin drivers, selecting all [57](#)
 - clock relationships, timing report [262](#)
 - clock report
 - asynchronous [259](#)
 - detailed [264](#)
 - Clock Tree, HDL Analyst tool [57](#)
 - clocks
 - asynchronous report [266](#)
 - declared clock [261](#)
 - defining [57](#)
 - derived clock [261](#)
 - inferred clock [261](#)
 - system clock [262](#)
 - Clocks panel, SCOPE [153](#)
 - collection commands
 - SCOPE [162](#)
 - Collections panel, SCOPE [161](#)
 - color coding
 - Text Editor [61](#)
 - commands
 - Tcl hooks [370](#)
 - commenting out code (Text Editor) [62](#)
 - compile points
 - Microsemi [419](#)
 - updating data (Microsemi) [413](#)
 - Compile Points panel, SCOPE [173](#)
 - compiler report, log file (.srr) [256](#)
 - compilers [24](#)
 - Constraint Check command [268](#)
 - constraint checking report [268](#)
 - constraint file
 - define_compile_point [238](#)
 - define_current_design [239](#)
 - constraint files [135](#)
 - .sdc [242](#)
 - automatic. *See* auto constraints
 - fdc and sdc precedence order [139](#)
 - Microsemi [408](#)
 - SCOPE spreadsheet [152](#)
 - SCOPE spreadsheet (Legacy) [198](#)
 - tcl script examples [373](#)
 - constraint files (.sdc)
 - creating [77](#)
 - constraint priority [139](#)
 - constraints
 - auto constraints. *See* auto constraints
 - FPGA timing [200](#)
 - non-DC [145](#)
 - priority [139](#)
 - report file [268](#)
 - styles [137](#)
 - types [134](#)
 - context help editor [63](#)
 - context of filtered schematic, displaying [123](#)
 - context sensitive help
 - using the F1 key [21](#)
 - copying
 - for pasting [85](#)
 - counter compiler
 - SYNCore [354](#)
 - create_clock timing constraint [202](#)
 - create_generated_clock timing constraint [204](#)
 - critical paths [128](#)
 - analyzing [129](#)
 - finding [129](#)
 - setting maximum (Microsemi) [413](#)
 - cross-clock paths, timing analysis [262](#)
 - cross-hair mouse pointer [73](#)
 - crossprobing [109](#)
 - definition [109](#)
 - Ctrl key
 - avoiding docking [76](#)
 - multiple selection [72](#)
 - zooming using the mouse wheel [74](#)
 - customization
 - callback functions [370](#)
 - cutting (for pasting) [77](#)
- ## D
- declared clock [261](#)
 - define_clock
 - forward-annotation, Microsemi [409](#)
 - define_compile_point
 - Tcl [238](#)

- define_current_design
 - Tcl [239](#)
- define_false_path
 - forward-annotation, Microsemi [409](#)
- define_multicycle_path
 - forward-annotation, Microsemi [409](#)
- define_path_delay
 - forward-annotation, Microsemi [409](#)
- defining I/O standards [171](#)
- delay paths
 - POS [191](#)
- Delay Paths panel, SCOPE [168](#)
- deleting
 - See removing
- derived clock [261](#)
- design flow
 - customizing with callback functions [370](#)
- design size, schematic sheet
 - setting [112](#)
- device options (Microsemi) [421](#)
- directives (Microsemi) [426](#)
- directory
 - examples delivered with synthesis tool [26](#)
- Dissolve Instances command [126](#)
- docking [45](#)
 - avoiding [76](#)
- docking GUI entities
 - toolbar [76](#)
- DSP blocks
 - inferencing [383](#)
- dual-port RAM examples [286](#)

E

- edf file [250](#)
- edif file (.edf) [250](#)
- editor view
 - context help [63](#)
- encoding
 - state machine
 - FSM Compiler [67](#)
 - FSM Explorer [69](#), [94](#)
- encryption
 - asymmetric [359](#)
 - methodologies [359](#)
 - symmetric [359](#)
- encryption algorithms [359](#)
- encryptIP script [364](#)
 - execution [365](#)
- encryptP1735 script [360](#)
 - multiple keys [362](#)
 - public keys [361](#)
- examples
 - Interactive Attribute Examples [64](#)
- examples delivered with synthesis tool, directory [26](#)
- Explorer, FSM
 - enabling [94](#)
 - overview [69](#)

F

- failures, timing (definition) [130](#)
- false paths
 - architectural [185](#)
 - clocks as from/to points [193](#)
 - code-introduced [185](#)
 - defined [185](#)
 - POS [191](#)
- fanout
 - Microsemi [410](#)
- FDC
 - create_clock constraint [202](#)
 - create_generated_clock [204](#)
 - reset_path [207](#)
 - set_clock_groups [209](#)
 - set_clock_latency [213](#)
 - set_clock_route_delay [215](#)
 - set_clock_uncertainty [216](#)
 - set_false_path [218](#)
 - set_input_delay [221](#)
 - set_max_delay [223](#)
 - set_multicycle_path [226](#)
 - set_output_delay [230](#)
 - set_reg_input_delay [233](#)
 - set_reg_output_delay [234](#)
- fdc
 - constraint priority [139](#)
 - precedence over sdc [139](#)
- fdc constraints [140](#)
 - generation process [138](#)
- fdc file
 - relationship with other constraint files [135](#)
- FIFO compiler
 - SYNCore [304](#)
- FIFO flags

- empty/almost empty [312](#)
- full/almost full [311](#)
- handshaking [313](#)
- programmable [314](#)
- programmable empty [318](#)
- programmable full [315](#)
- files
 - .adc [242](#)
 - .areasrr [249](#)
 - .edf [250](#)
 - .fdc [242](#)
 - .fse [249](#)
 - .info [249](#)
 - .ini [242](#)
 - .prj [22](#), [242](#)
 - .sar [251](#)
 - .sdc [242](#)
 - .srm [251](#), [252](#)
 - .srr [254](#)
 - watching selected information [44](#)
 - .srs [252](#)
 - .ta [252](#)
 - .v [243](#)
 - .vhd [243](#)
 - .vhm [254](#)
 - .vm [254](#)
 - compiler output (.srs) [252](#)
 - constraint (.adc) [242](#)
 - constraint (.sdc) [242](#)
 - creating [77](#)
 - customized timing report (.ta) [252](#)
 - design component info (.info) [249](#)
 - edif (.edf) [250](#)
 - initialization (.ini) [242](#)
 - log (.srr) [254](#)
 - watching selected information [44](#)
 - mapper output (.srm) [251](#), [252](#)
 - output
 - See output files
 - project (.prj) [22](#), [242](#)
 - RTL view (.srs) [252](#)
 - srr [254](#)
 - watching selected information [44](#)
 - state machine encoding (.fse) [249](#)
 - Synopsys archive file (.sar) [251](#)
 - synthesis output [249](#)
 - Technology view (.srm) [251](#), [252](#)
 - Verilog (.v) [243](#)
 - VHDL (.vhd) [243](#)
- files for synthesis [242](#)
- filtered schematic
 - compared with unfiltered [96](#)
- filtering [122](#)
 - commands [122](#)
 - compared with flattening [126](#)
 - FSM states and transitions [59](#)
 - paths from pins or ports [130](#)
- filtering critical paths [129](#)
- finding
 - critical paths [129](#)
 - information on synthesis tool [28](#)
 - GUI [21](#)
- finite state machines
 - See state machines
- Flatten Current Schematic command [126](#)
- Flatten Schematic command [126](#)
- flattening
 - commands [124](#)
 - compared with filtering [126](#)
 - selected instances [125](#)
- Float command
 - Watch window popup menu [45](#)
- floating
 - toolbar [76](#)
- floating toolbar popup menu [76](#)
- forward annotation
 - initial values [297](#)
- Forward Annotation of Initial Values
 - Verilog [297](#)
- forward-annotation
 - Microsemi [408](#)
- FPGA timing constraints [200](#)
- frequency
 - cross-clock paths [263](#)
- Frequency (Mhz) option, Project view [93](#)
- from points
 - clocks [192](#)
 - multiple [188](#)
 - objects [187](#)
- fse file [249](#)
- FSM Compiler option, Project view [93](#)
- FSM Compiler, enabling and disabling
 - globally
 - with GUI [93](#)
 - locally, for specific register [68](#)
- FSM encoding file (.fse) [249](#)
- FSM Explorer
 - enabling [94](#)

- overview [69](#)
- FSM Explorer option, Project view [94](#)
- FSM toolbar [81](#)
- FSM Viewer [58](#)
- FSMs (finite state machines)
 - See state machines

G

- Generated Clocks panel, SCOPE [159](#)
- generic technology library [246](#)
- graphical user interface (GUI), overview [29](#)
- GTECH library. See generic technology library
- gtech.v library [246](#)
- gui
 - synthesis software [19](#)
- GUI (graphical user interface), overview [29](#)

H

- HDL Analyst tool [95](#)
 - accessing commands [97](#)
 - analyzing critical paths [128](#)
 - Clock Tree [57](#)
 - crossprobing [109](#)
 - filtering designs [122](#)
 - finding objects [107](#)
 - hierarchical instances. See hierarchical instances
 - object information [98](#)
 - preferences [112](#)
 - push/pop mode [115](#)
 - ROM table viewer [298](#)
 - schematic sheet size [112](#)
 - schematics, filtering [122](#)
 - schematics, multiple-sheet [112](#)
 - status bar information [98](#)
 - title bar information [112](#)
- HDL Analyst toolbar
 - See Analyst toolbar
- HDL Analyst views [96](#)
 - See also RTL view, Technology view
- HDL files, creating [77](#)
- header, timing report [260](#)
- help
 - online
 - accessing [21](#)
- hidden hierarchical instances [103](#)

- are not flattened [126](#)
- Hide command
 - floating toolbar popup menu [76](#)
 - Log Watch window popup menu [45](#)
 - Tcl Window popup menu [48](#)
- hierarchical area report [267](#)
 - .areasrr file [267](#)
- hierarchical instances [101](#)
 - compared with primitive [100](#)
 - display in HDL Analyst [101](#)
 - hidden [103](#)
 - opaque [101](#)
 - transparent [101](#)
- hierarchical project management views [33](#)
- hierarchical schematic sheet, definition [112](#)
- hierarchy
 - flattening
 - compared with filtering [126](#)
 - pushing and popping [115](#)
 - schematic sheets [112](#)
- Hierarchy Browser [118](#)
 - changing size in view [53](#)
 - Clock Tree [57](#)
 - finding schematic objects [107](#)
 - moving between objects [57](#)
 - RTL view [53](#)
 - symbols (legend) [57](#)
 - Technology view [55](#)
 - trees of objects [56](#)
- hierarchy separator [235](#)

I

- I/O constraints
 - multiple on same port [166](#)
- I/O insertion (Microsemi) [412](#)
- I/O Standards panel, SCOPE [171](#)
- Identify Instrumentor
 - launching [83](#)
- IEEE 1364 Verilog 95 standard [244](#)
- Implementation Directory [39](#)
- Implementation Results [39](#)
- indenting a block of text [61](#)
- indenting text (Text Editor) [61](#)
- inferencing
 - DSP blocks [383](#)
- inferred clock [261](#)
- info file (design component info) [249](#)
- ini file [242](#)

initial value data file

Verilog 294

Initial Values

forward annotation 297

initial values

\$readmemb 292

\$readmemh 292

initial values (Verilog)

netlist file (.srs) 297

initialization file (.ini) 242

input files 242

.adc 242

.ini 242

.sdc 242

.sv 243

.v 243

.vhd 243

Inputs/Outputs panel, SCOPE 163

inserting

bookmarks (Text Editor) 61

instances

hierarchical

dissolving 120

making transparent 120

hierarchical. *See* hierarchical instances

primitive. *See* primitive instances

Interactive Attribute Examples 64

interface information, timing report 263

isolating paths from pins or ports 130

K

keyboard shortcuts 84

arrow keys (Hierarchy Browser) 119

keyword completion, Text Editor 61

keywords

completing in Text Editor 61

L

latches

in timing analysis 128

Launch Identify Instrumentor icon 83

legacy sdc file. *See* sdc files, difference between legacy and Synopsys standard

lib2syn

using 247

libraries

general technology 245

macro, built-in 243

technology-independent 245

VHDL

attributes and constraints 244

license

specifying in batch mode 23

linkerlog file 250

log file (.srr) 254

watching selected information 44

log file report 254

clock buffering 256

compiler 256

mapper 256

net buffering 257

resource usage 258

retiming 258

summary of compile points 257

timing 257

Log Watch Configuration dialog box 46

Log Watch window 44

Output Windows 52

positioning commands 45

M

macros

libraries 243

MATH18X18 block 382

Microsemi 381

SIMBUF 382

mapper output file (.srm) 251, 252

mapper report

log file (.srr) 256

margin, slack 129

message viewer

description 48

Messages Tab 48

Microsemi

attributes 426

black boxes 381

compile point synthesis 419

compile point timing data 413

device options 421

directives 426

features 379

forward-annotation, constraints 408

I/O insertion 412

macros 381

MATH18X18 block 382

Operating Condition Device Option 415

- product families 378
- reports 409
- retiming 413
- SIMBUF macro 382
- Tcl implementation options 423
- Microsemi implementing RAM 387
- mouse button operations 72
- mouse operations 69
- Mouse Stroke Tutor 71
- mouse wheel operations 74
- Move command
 - floating toolbar window 76
 - Log Watch window popup menu 45
 - Tcl window popup menu 48
- moving between objects in the Hierarchy
 - Browser 57
- moving GUI entities
 - toolbar 76
- multicycle paths
 - clocks as from/to points 192
 - examples 183
 - POS 191
 - using different start/end clocks 182
- multiple target technologies, synthesizing
 - with Tcl script 372
- multiple-sheet schematics 112
- multipliers
 - DSP blocks 383
- multisheet schematics
 - transparent hierarchical instances 114

N

- naming rules 234
- navigating
 - among hierarchical levels
 - by pushing and popping 115
 - with the Hierarchy Browser 118
 - among the sheets of a schematic 112
- nesting design details (display) 120
- net buffering report, log file 257
- netlist file 254
 - initial values (Verilog) 297

O

- object information
 - status bar, HDL Analyst tool 98
 - viewing in HDL Analyst tool 98

- objects
 - crossprobing 109
 - dissolving 120
 - making transparent 120
- objects, schematic
 - See schematic objects
- Online help
 - F1 key 21
- online help
 - accessing 21
- opaque hierarchical instances 101
 - are not flattened 126
- optimization
 - state machines 67
- options
 - Project view 92
 - Frequency (Mhz) 93
 - FSM Compiler 93
 - FSM Explorer 94
 - Resource Sharing 94
 - Retiming 94
 - setting with set_option Tcl command 373
- options (Microsemi) 423
- output files 249
 - .areasrr 249
 - .edf 250
 - .info 249
 - .sar 251
 - .srm 251, 252
 - .srr 254
 - watching selected information 44
 - .srs 252
 - .ta 252
 - .vhm 254
 - .vm 254
 - netlist 254
 - See also files
- Output Windows 52
- Overview of the Synopsys FPGA Synthesis Tools 18

P

- partitioning of schematics into sheets 112
- pasting 77
- path delays
 - clocks as from/to points 193
- performance summary, timing report 260

- pins
 - displaying
 - on transparent instances [105](#)
 - displaying on technology-specific primitives [106](#)
 - isolating paths from [130](#)
- Place and Route constraint file (Microsemi) [408](#)
- pointers, mouse
 - cross-hairs [73](#)
 - push/pop arrows [117](#)
- popping up design hierarchy [115](#)
- popup menus
 - floating toolbar [76](#)
 - Log Watch window [45](#), [46](#)
 - Log Watch window positioning [45](#)
 - Tcl window [48](#)
- POS
 - interface [190](#)
- precedence of constraint files [139](#)
- preferences
 - HDL Analyst tool [112](#)
 - Project view display [74](#)
- primitive instances [100](#)
- primitives
 - pin names in Technology view [106](#)
- private key [360](#)
- prj file [22](#), [242](#)
- Process View [40](#)
- Product of Sums
 - See POS
- project files (.prj) [22](#), [242](#)
- project results
 - Implementation Directory [39](#)
 - Process View [40](#)
 - Project Status View [34](#)
- Project Results View [34](#)
- Project Status View [34](#)
- Project toolbar [76](#)
- Project view [30](#)
 - buttons and options [92](#)
 - options [92](#)
 - Synplify Pro [30](#)
- Project window [30](#)
- project_name_cck.rpt file [268](#)
- Promote Global Buffer Threshold (Microsemi) [411](#)
- public key [359](#)

- push/pop mode, HDL Analyst tool [115](#)

R

- RAM implementations
 - Microsemi [387](#)
- RAMs
 - initial values (Verilog) [292](#)
- RAMs, inferring
 - advantages [278](#)
- Registers panel, SCOPE [167](#)
- removing
 - bookmark (Text Editor) [61](#)
 - window (view) [76](#)
- reports
 - constraint checking (cck.rpt) [268](#)
 - reset_path timing constraint [207](#)
 - resolving conflicting timing constraints [194](#)
 - Resource Sharing option, Project view [94](#)
 - resource usage report, log file [258](#)
- retiming
 - report, log file [258](#)
- retiming (Microsemi) [413](#)
- Retiming option, Project view [94](#)
- ROM compiler
 - SYNCore [336](#)
- ROM inference examples [298](#)
- ROM initialization
 - with rom.info file [301](#)
 - with Verilog generate block [302](#)
- rom.info file [298](#)
- RTL view [53](#)
 - displaying [79](#)
 - file (.srs) [252](#)

S

- schematic objects
 - crossprobing [109](#)
 - definition [98](#)
 - dissolving [120](#)
 - finding [107](#)
 - making transparent [120](#)
 - status bar information [98](#)
- schematic sheets [112](#)
 - hierarchical (definition) [112](#)
 - navigating among [112](#)
 - setting size [112](#)
- schematics

- configuring amount of logic on a sheet 112
- crossprobing 109
- filtered 96
- filtering commands 122
- flattening compared with filtering 126
- flattening selectively 125
- hierarchical (definition) 112
- multiple-sheet 112
- multiple-sheet. *See also* schematic sheets
- object information 98
- partitioning into sheets 112
- sheet connectors 99
- sheets
 - navigating among 112
 - size, setting 112
- size in view, changing 53
- unfiltered 96
- unfiltering 123
- SCOPE
 - Attributes panel 170
 - clock groups 155
 - Clocks panel 153
 - Collections panel 161
 - Compile Points panel 173
 - Delay Paths panel 168
 - for legacy sdc 142
 - Generated Clocks panel 159
 - I/O Standards panel 171
 - Inputs/Outputs panel 163
 - Registers panel 167
 - TCL View 176
- SCOPE spreadsheet
 - starting 152
- SCOPE timing constraints summary 153
- sdc
 - fdc precedence 139
 - SCOPE for legacy files 142
- sdc file
 - difference between legacy and Synopsys standard 137
- sdc2fdc utility 143
- Search SolvNet
 - using 66
- selecting
 - text column (Text Editor) 62
- selecting multiple objects using the Ctrl key 72
- sequential elements
 - naming 235
- set modules command (collections) 162
- set modules_copy command (collections) 162
- set_clock_groups timing constraint 209
- set_clock_latency timing constraint 213
- set_clock_route_delay timing constraint 215
- set_clock_uncertainty timing constraint 216
- set_false_path timing constraint 218
- set_hierarchy_separator command 235
- set_input_delay timing constraint 221
- set_max_delay timing constraint 223
- set_multicycle_path timing constraint 226
- set_output_delay timing constraint 230
- set_reg_input_delay timing constraint 233
- set_reg_output_delay timing constraint 234
- set_rtl_ff_names 145
- set_rtl_ff_names command 235
- sheet connectors 99
- Shift key 76
- shortcuts
 - keyboard
 - See* keyboard shortcuts
- SIMBUF macro 382
- single-port RAM examples 283
- slack
 - cross-clock paths 263
 - defined 261
 - margin
 - definition 130
 - setting 129
- SolvNet
 - search 66
- source files
 - See also* files
 - creating 77
- srd file 251
- srm file 251, 252
- srr file 254
 - watching selected information 44
- srs file 252
 - initial values (Verilog) 297
- standards, supported
 - Verilog 244
 - VHDL 244
- starting Synplify 23
- starting Synplify Pro 23
- state machines

- encoding
 - displaying [59](#)
 - FSM Compiler [67](#)
 - FSM Explorer [69](#), [94](#)
 - encoding file (.fse) [249](#)
 - filtering states and transitions [59](#)
 - optimization [67](#)
 - state encoding, displaying [59](#)
 - status bar information, HDL Analyst tool [98](#)
 - structural netlist file (.vhm) [254](#)
 - structural netlist file (.vm) [254](#)
 - subtractor
 - SYNCore [342](#)
 - summary of compile points report
 - log file (.srr) [257](#)
 - supported standards
 - Verilog [244](#)
 - VHDL [244](#)
 - symbols
 - Hierarchy Browser (legend) [57](#)
 - syn_maxfan
 - fanout limits (Microsemi) [410](#)
 - syn_reference_clock attribute
 - effect on multiple I/O constraints [166](#)
 - SYN_TCL_HOOKS variable [370](#)
 - SYNCore
 - adder/subtractor [342](#)
 - byte-enable RAM compiler
 - byte-enable RAM compiler
 - SYNCore [331](#)
 - counter compiler [354](#)
 - FIFO compiler [304](#)
 - RAM compiler
 - RAM compiler
 - SYNCore [321](#)
 - ROM compiler [336](#)
 - SYNCore adder/subtractor
 - adders [343](#)
 - dynamic adder/subtractor [349](#)
 - functional description [342](#)
 - subtractors [346](#)
 - SYNCore FIFOs
 - definition [304](#)
 - parameter definitions [309](#)
 - port list [307](#)
 - read operations [306](#)
 - status flags [311](#)
 - write operations [305](#)
 - SYNCore ROMs
 - clock latency [340](#)
 - dual-port read [338](#)
 - parameter list [339](#)
 - single-port read [338](#)
 - synhooks.tcl file [370](#)
 - Synopsys FPGA Synthesis Tools
 - overview [18](#)
 - Synopsys standard sdc file. *See* sdc files, difference between legacy and Synopsys standard
 - Synplify Pro synthesis tool
 - overview [14](#)
 - Synplify Pro tool
 - Project view [30](#)
 - user interface [19](#)
 - synplify_pro command-line command [23](#)
 - syntax
 - bus dimension separator [236](#)
 - bus naming [236](#)
 - synthesis
 - log file (.srr) [254](#)
 - watching selected information [44](#)
 - synthesis software
 - flow [24](#)
 - gui [19](#)
 - system clock [262](#)
 - SystemVerilog keywords
 - context help [63](#)
- ## T
- ta file (customized timing report) [252](#)
 - Tcl commands
 - collections [162](#)
 - constraint files [141](#)
 - pasting [48](#)
 - syntax for Tcl hooks [370](#)
 - Tcl Script window
 - Output Windows [52](#)
 - Tcl scripts
 - examples [372](#)
 - Tcl shell command
 - sdc2fdc [143](#)
 - TCL View, SCOPE [176](#)
 - Tcl window
 - popup menu commands [48](#)
 - popup menus [48](#)
 - Technology view [54](#)
 - displaying [79](#)

- file (.srm) 251, 252
 - Text Editor
 - features 61
 - indenting a block of text 61
 - opening 60
 - selecting text column 62
 - view 60
 - text editor
 - completing keywords 61
 - Text Editor view 60
 - through constraints
 - point-to-point delays 168
 - through points
 - clocks 193
 - lists, multiple 190
 - lists, single 189
 - multiple 190
 - product of sums UI 190
 - single 189
 - specifying for timing exceptions 189
 - timing analysis of critical paths (HDL Analyst tool) 128
 - timing analyst
 - cross-clock paths 262
 - timing annotated properties (.tap) 253
 - timing constraints
 - conflict resolution 194
 - constraint priority 194
 - create_clock 202
 - create_generated_clock 204
 - FPGA 200
 - reset_path 207
 - See also* FPGA timing constraints
 - See* constraints
 - set_clock_groups 209
 - set_clock_latency 213
 - set_clock_route_delay 215
 - set_clock_uncertainty 216
 - set_false_path 218
 - set_input_delay 221
 - set_max_delay 223
 - set_multicycle_path 226
 - set_output_delay 230
 - set_reg_input_delay 233
 - set_reg_output_delay 234
 - timing exceptions
 - False Paths panel 185
 - multicycle paths 182
 - priority 194
 - specifying paths/points 185
 - timing failures, definition 130
 - timing report 259
 - clock relationships 262
 - customized (.ta file) 252
 - detailed clock report 264
 - file (.ta) 252
 - header 260
 - interface information 263
 - performance summary 260
 - timing reports
 - asynchronous clocks 266
 - log file (.srr) 257
 - title bar information, HDL Analyst tool 112
 - to points
 - clocks 192
 - multiple 188
 - objects 187
 - toolbars 76
 - FSM 81
 - moving and docking 76
 - transparent hierarchical instances 102
 - lower-level logic on multiple sheets 114
 - operations resulting in 121
 - pins and pin names 105
 - trees of objects, Hierarchy Browser 56
 - trees, browser, collapsing and expanding 57
- ## U
- unfiltered schematic, compared with filtered 96
 - unfiltering schematic 123
 - user interface
 - Synplify Pro tool 19
 - user interface, overview 29
 - using the mouse 69
 - utilities
 - lib2syn 247
 - sdcd2fdc 143
- ## V
- v file 243
 - vendor technologies
 - Microsemi 377
 - vendor-specific Tcl commands 370
 - Verilog
 - Forward Annotation of Initial Values 297

- generic technology library [246](#)
- initial value data file [294](#)
- initial values for RAMs [292](#)
- netlist file [254](#)
- ROM inference [298](#)
- source files (.v) [243](#)
- structural netlist file (.vm) [254](#)
- supported standards [244](#)
- Verilog 2001 [244](#)
- Verilog 95 [244](#)
- Verilog source file (.v) [243](#)
- vhd file [243](#)
- vhd source file [243](#)
- VHDL
 - libraries
 - attributes, supplied with synthesis tool [244](#)
 - source files (.vhd) [243](#)
 - structural netlist file (.vhm) [254](#)
 - supported standards [244](#)
- VHDL source file (.vhd) [243](#)
- vhm file [254](#)
- views [43](#)
 - FSM [58](#)
 - managing [75](#)
 - Project [30](#)
 - removing [76](#)
 - RTL [53](#)
 - Technology [54](#)
- vm file [254](#)

W

- Watch Window. *See* Log Watch window
- window
 - Project [30](#)
- windows [43](#)
 - closing [86](#)
 - log watch [44](#)
 - removing [76](#)

Z

- zoom
 - using the mouse wheel and Ctrl key [74](#)