



Synplicity®

Simply Better Results

Synplicity® FPGA Synthesis

*Synplify®, Synplify Pro®, Synplify® Premier, and
Synplify® Premier with Design Planner*

**User Guide
December 2005**

Synplicity, Inc.
600 West California Avenue
Sunnyvale, CA 94086
(U.S.) +1 408 215-6000 direct
(U.S.) +1 408 222-0268 fax
www.synplicity.com

Contents

Chapter 1: Introduction

The FPGA Synthesis Tools	1-2
About the Synplify and Synplify Pro Software	1-2
About the Synplify Premier Software	1-3
Synplicity FPGA Tool Features	1-4
Synplicity Product Family	1-8
The Generic FPGA Design Flow	1-8
HDL Design Entry	1-9
Logic Optimization (Compilation)	1-9
Technology Mapping	1-10
Placement	1-10
Routing	1-10
FPGA Configuration	1-10
Audience	1-11
Scope of the Document	1-11
Starting the Software	1-12
Getting Started	1-12
Getting Help	1-13
User Interface Overview	1-14
Design Flows	1-16
Logic Synthesis Design Flow	1-16
Prototyping Design Flow	1-17
Physical Synthesis Design Flows	1-17

Chapter 2: Project Setup

Setting Up HDL Source Files	2-2
Creating HDL Source Files	2-2
Checking HDL Source Files	2-4
Editing HDL Source Files with the Built-in Text Editor	2-5

Using an External Text Editor	2-8
Setting Editing Window Preferences	2-9
Setting Up Project Files	2-11
Creating a Project File	2-11
Opening an Existing Project File	2-14
Making Changes to a Project	2-15
Using Mixed Language Source Files	2-16
Setting Project View Display Preferences	2-18
Updating Verilog Include Paths in Older Project Files	2-21
Setting Up Implementations and Workspaces	2-22
Working with Multiple Implementations	2-22
Creating Workspaces	2-24
Using Workspaces	2-25
Archiving Files and Projects	2-26

Chapter 3: Constraints, Attributes, and Options

Setting Implementation Options	3-2
Setting Device Options	3-2
Setting Optimization Options	3-5
Specifying Global Frequency and Constraint Files	3-6
Specifying Result Options	3-9
Specifying Timing Report Output	3-10
Setting Verilog and VHDL Options	3-11
Setting Synplify Premier Netlist Restructuring Optimizations	3-16
Setting Constraints in the SCOPE Window	3-18
Using the SCOPE Window	3-19
Entering and Editing Constraints in the SCOPE Window	3-21
Entering Default Constraints	3-24
Setting Clock and Path Constraints	3-26
Defining Clocks	3-28
Defining Input and Output Constraints	3-33
Defining Multicycle Paths	3-34
Defining From/To/Through for Timing Exceptions	3-35
Defining False Paths	3-38
Specifying Standard I/O Pad Types	3-40
Setting SCOPE Display Preferences	3-41
Translating Xilinx UCF Constraints	3-42
Converting Pin Location Constraint Files in the Synplify Premier Tool	3-43

Using Auto Constraints	3-45
Using Collections	3-47
Comparing Methods for Defining Collections	3-47
Creating and Using Collections (SCOPE Window)	3-48
Creating Collections (Tcl Commands)	3-51
Using the Tcl Find Command to Define Collections	3-53
Using the Expand Tcl Command to Define Collections	3-55
Viewing and Manipulating Collections (Tcl Commands)	3-56
Working with Constraint Files	3-60
When to Use Constraint Files over Source Code	3-60
Tcl Syntax Guidelines for Constraint Files	3-61
Using a Text Editor for Constraint Files	3-62
Generating Constraint Files for Forward Annotation	3-64
Adding Attributes and Directives	3-66
Adding Attributes and Directives in VHDL	3-66
Adding Attributes and Directives in Verilog	3-68
Adding Attributes in the SCOPE Window	3-68
Adding Attributes with the SCOPE Wizard	3-71
Adding Attributes to a Tcl Constraint File	3-73
Adding Attributes From the RTL and Technology Views	3-74

Chapter 4: Result Analysis

Checking Log Results	4-2
Viewing the Log File	4-2
Analyzing Results Using the Log File Reports	4-5
Using the Log Watch Window	4-6
Handling Messages	4-8
Checking Results in the Message Viewer	4-8
Filtering Messages in the Message Viewer	4-10
Filtering Messages from the Command Line	4-13
Handling Warnings	4-14
Automating Message Filtering with a synhooks Script	4-14
Basic Operations in the Schematic Views	4-16
Differentiating Between the Views	4-17
Opening the Views	4-17
Analyzing Your Design Graphically	4-19
Viewing Object Properties	4-20
Selecting Objects in the RTL/Technology Views	4-23
Working with Multisheet Schematics	4-24
Moving Between Views in a Schematic Window	4-26

Setting Schematic View Preferences	4-26
Managing Windows	4-28
Exploring Design Hierarchy	4-30
Traversing Design Hierarchy with the Hierarchy Browser	4-30
Exploring Object Hierarchy by Pushing/Popping	4-31
Exploring Object Hierarchy of Transparent Instances	4-36
Finding Objects	4-37
Browsing to Find Objects	4-37
Using Find for Hierarchical and Restricted Searches	4-39
Using Wildcards with the Find Command	4-42
Using Find to Search the Output Netlist	4-45
Crossprobing	4-48
Crossprobing Description	4-48
Crossprobing within an RTL/Technology View	4-49
Crossprobing from the RTL/Technology View	4-49
Crossprobing from the Text Editor Window	4-51
Crossprobing from the Tcl Script Window	4-54
Crossprobing from the FSM Viewer	4-54
Analyzing With the HDL Analyst Tool	4-56
Viewing Design Hierarchy and Context	4-56
Filtering Schematics	4-60
Expanding Pin and Net Logic	4-62
Expanding and Viewing Connections	4-66
Flattening Schematic Hierarchy	4-67
Minimizing Memory Usage While Analyzing Designs	4-72
Analyzing Timing	4-73
Analyzing Clock Trees in the RTL View	4-73
Viewing Critical Paths	4-74
Analyzing Paths with the Timing Analyst	4-76
Analyzing Paths with the Synplify Premier Timing Analyst	4-79
Handling Negative Slack	4-81
The Island Timing Report	4-83
Generating the Island Timing Report	4-83
Automatic Island Timing Report	4-84
Defining the Group Range and Global Range	4-85
Interactive Island Timing Analyst	4-86
Viewing the Island Timing Report	4-87

The Island Timing Analyst	4-88
Islands/Paths Control Panel	4-89
Islands/Paths Summary View	4-92
Islands/Paths Summary Management	4-93
Islands/Paths Details View	4-95
Island Timing Report Critical Paths	4-98
Assigning Critical Paths to a Region	4-98

Chapter 5: Physical Analyst

Synplify Premier Physical Analyst Tool	5-2
Opening the Physical Analyst View	5-4
Using the Physical Analyst Control Panel	5-5
Setting Object Controls	5-6
Setting Visibility Controls	5-6
Using the Physical Analyst Device View	5-8
Setting Object Display Options	5-10
Displaying Instances	5-10
Displaying Signal Pins	5-10
Displaying Signal Flow for Selected Nets	5-11
Routing Nets to Display	5-12
Selecting Objects	5-14
Selecting Multiple Nets	5-15
Transcribing Object Selections	5-16
Viewing Object Information	5-17
Viewing Properties	5-17
Using Tool Tips	5-20
Using Mouse Strokes	5-20
Using Keyboard Shortcuts	5-21
Zooming in the Physical Analyst	5-22
Finding Objects	5-23
Finding Objects with the Find Command	5-23
Finding Object Locations	5-27
Using Markers	5-29
Changing Color Schemes	5-31
Configuring Enhanced Instance Display	5-31
Crossprobing in Physical Analyst	5-34

Retiming	6-44
Controlling Retiming	6-44
Retiming Example	6-46
Retiming Report	6-48
How Retiming Works	6-48
How Retiming Works With Synplify Premier Regions	6-50
Inserting Probes	6-50
Specifying Probes in the Source Code	6-51
Adding Probe Attributes Interactively	6-52
Inferring RAMs	6-54
Inference vs. Instantiation	6-54
Coding RAMs for Inference	6-55
Specifying RAM Implementation Styles	6-59
Implementing Altera RAMs Automatically	6-61
Implementing Xilinx RAMs Automatically	6-64
Implementing Altera RAMs: FLEX and APEX	6-67
Implementing Altera RAMs: Stratix Multi-Port RAMs	6-69
Inferring Xilinx Block RAMs Using Registered Addresses	6-70
Inferring Xilinx Block RAMs Using Registered Output	6-73
Setting Xilinx RAM Initialization Values	6-78
Mapping Xilinx ROM to Block RAM	6-79
Inferring Shift Registers	6-80
Shift Register Examples	6-82
Forward Annotation of Initial Values	6-86
Working with LPMs	6-87
Instantiating LPMs as Black Boxes (Altera)	6-88
Instantiating LPMs as Black Boxes (Cypress)	6-92
Instantiating LPMs Using VHDL Prepared Components	6-94
Instantiating LPMs Using a Verilog Library (Altera)	6-97
Working with Gated Clocks	6-99
The Synplicity Approach to Gated Clocks	6-99
Synthesizing a Gated Clock Design	6-101
Prerequisites for Gated Clock Conversion	6-103
Gated Clock Conversion Report	6-105
Fix Gated Clock Error Messages	6-106
Gated Clocks for Black Boxes	6-108
Restrictions to Using Fix Gated Clocks	6-110
Generated-Clock Optimization	6-111

Chapter 7: Design Planning and Optimizations

Using the Design Planner	7-2
Creating a Design Plan	7-2
Cutting, Copying, and Pasting in the Design Planner	7-5
Pin Assignments	7-6
Methods for Specifying Pin Assignments	7-6
Specifying Pins Using the Design Plan Editor	7-7
Implementing Pin Assignments	7-11
Storing Temporary Pin Assignments	7-14
Displaying Rats Nesting	7-15
Pin Assignment Statistics	7-16
Assigning Clock Pins	7-17
Working with Regions	7-19
Viewing Intellectual Property (IP) Core Areas	7-19
Placing Regions	7-20
Moving and Sizing Regions	7-21
Replicating Logic Manually	7-23
Assigning Register to Pin-Locked I/O Paths to Regions	7-24
Checking Synplify Premier Utilization	7-25
Device Utilization	7-25
Region Utilization	7-25
Using Process-Level Hierarchy	7-25
Bit Slicing	7-26
About Bit Slicing	7-26
Using Bit Slicing	7-26
Bit Slice Examples	7-29
Bit Slicing Guidelines	7-32
Zippering	7-33
Using Zippering	7-33
Analyzing a Design for Zippering	7-36
Zippering Example	7-37
Zippering Guidelines	7-41

Analyzing Netlist with the Physical Analyst	5-42
Expanding and Viewing Connections	5-48
Analyzing Timing with the Physical Analyst	5-51
Viewing Critical Paths	5-51
Tracing Critical Paths Forward and Backwards	5-54

Chapter 6: Design Optimization

Design Guidelines	6-2
General Optimization Tips	6-2
Area Optimization Tips	6-3
Timing Optimization Settings	6-4
Optimizing Results	6-5
Sharing Resources	6-5
Setting Fanout Limits	6-7
Controlling Buffering and Replication	6-8
Controlling Hierarchy Flattening	6-10
Preserving Objects from Optimization	6-10
Preserving Hierarchy	6-12
Defining State Machines for Synthesis	6-13
Defining State Machines in Verilog	6-13
Defining State Machines in VHDL	6-14
Specifying FSMs with Attributes and Directives	6-15
Using the Symbolic FSM Compiler	6-17
Choosing When to Use the FSM Compiler	6-17
Running the FSM Compiler on the Whole Design	6-18
Running the FSM Compiler on Individual FSMs	6-20
Using FSM Explorer	6-22
Deciding When to Use the FSM Explorer	6-22
Running the FSM Explorer	6-23
Using the FSM Viewer	6-25
Defining Black Boxes for Synthesis	6-30
Instantiating Black Boxes and I/Os in Verilog	6-30
Instantiating Black Boxes and I/Os in VHDL	6-32
Adding Black Box Timing Constraints	6-34
Adding Other Black Box Attributes	6-38
Pipelining	6-40
Prerequisites for Pipelining	6-40
Pipelining the Design	6-41

Chapter 8: Vendor-Specific Optimizations

Passing Information to the P&R Tools	8-2
Specifying Pin Locations	8-2
Specifying Locations for Actel Bus Ports	8-3
Specifying Macro and Register Placement	8-3
Passing Technology Properties	8-4
Specifying Padtype and Port Information	8-5
Generating Vendor-Specific Output	8-6
Targeting Output to Your Vendor	8-6
Customizing Netlist Formats	8-7
Working with Actel Designs	8-8
Using Predefined Actel Black Boxes	8-8
Using ACTGen Macros	8-9
Working with Radhard Designs	8-10
Working with Altera Designs	8-11
APEX Design Tips	8-12
FLEX Design Tips	8-12
Determining ROM Implementation	8-12
Working with Altera EABs and ESBs	8-14
Working with Altera PLLs	8-15
Implementing Megafunctions with Clearbox	8-16
Packing I/O Cell Registers	8-18
Using LPMs in Simulation Flows	8-20
Working with Quartus II	8-22
Working with Lattice Designs	8-23
Instantiating Lattice Macros	8-23
Using Lattice GSR Resources	8-24
Inferring Carry Chains in Lattice XPLD Devices	8-25
Controlling I/O Insertion in Lattice Designs	8-25
Forward-Annotating Lattice ORCA Constraints	8-26
Working with Xilinx Designs	8-28
Designing for Xilinx Architectures	8-28
Instantiating CoreGen Cores	8-29
Instantiating Virtex PCI Cores	8-30
Packing Registers for I/Os	8-33
Controlling Placement with RLOCs	8-35
Using Clock Buffers in Virtex Designs	8-36
Instantiating Special I/O Standard Buffers for Virtex	8-38
Reoptimizing With EDIF Files	8-39
Working with Xilinx Place-and-Route Software	8-39

Chapter 9: Design Planning for Vendors

Design Planning with Altera Devices	9-2
Stratix and Cyclone Devices	9-2
Displaying Stratix Devices	9-2
Creating Regions for Stratix Devices	9-4
Design Planning with Xilinx Designs	9-8
Displaying Xilinx Device Resources	9-8
Creating Regions for Xilinx Designs	9-12
Handling Xilinx Critical Paths (Design Planner)	9-14
Splitting a Critical Path into Multiple Regions	9-14
Creating Smaller Regions for Long Critical Paths	9-15
Handling Critical Paths with High Fanout Nets	9-15
Handling Critical Paths with Cascading Cells or Carry Chain Logic	9-17
Handling Critical Paths with Bit Slicing	9-19
Handling Critical Paths with Pipelining	9-20
Handling Designs with Multiple Critical Paths	9-20
Handling Critical Paths with Large Multiplexers	9-21
Handling Xilinx Black Boxes (Design Planner)	9-22
Design Planning Xilinx Black Boxes	9-22
Handling Xilinx Block RAMs (Design Planner)	9-24
Creating Block RAM Regions	9-25
Assigning to Block RAM Regions	9-27
Handling Block Multipliers (Design Planner)	9-30
Block Multiplier Support	9-30
Creating Block Mult Regions	9-30
Assigning to Block Mult Regions	9-31
Region Utilization	9-31
Handling DSP Blocks (Design Planner)	9-32
Handling Xilinx IPs (Design Planner)	9-35
Intrusive IP Flow	9-35
Macro IP Flow	9-37
Creating IP Region Constraints	9-38

Chapter 10: Design Flows and Process Optimization

Using Batch Mode	10-2
Running Batch Mode on a Project File	10-2
Running Batch Mode with a Tcl Script	10-3
Working with Tcl Scripts and Commands	10-4
Crossprobing from the Tcl Script Window	10-4
Using Tcl Commands and Scripts	10-4
Generating a Job Script	10-5
Creating a Tcl Synthesis Script	10-5
Using Tcl Variables to Try Different Clock Frequencies	10-7
Using Tcl Variables to Try Several Target Technologies	10-8
Running Bottom-up Synthesis with a Script	10-9
Automating Flows with synhooks.tcl	10-10
The VIF Formal Verification Flow	10-13
Overview of the VIF Flow	10-13
Generating a VIF File	10-14
Using a Tcl Script for VIF Conversion	10-16
Handling Equivalency Check Failures	10-18
Protected Flow Support	10-19
Using IP in a Design	10-20
Running Place-and-Route After Synthesis	10-21
Creating and Running P&R Projects	10-21
Specifying Xilinx Place-and-Route Options	10-23
Backannotating Place-and-Route Data	10-25
Analyzing Physical Synthesis (Synplify Premier)	10-26
MultiPoint Synthesis	10-28
Traditional Bottom-up Design and MultiPoint Synthesis	10-28
The Synplify Pro MultiPoint Synthesis Flow	10-29
The Altera LogicLock Flow	10-39
Using Synplify Pro With the Altera LogicLock Flow	10-39
Using Synplify Premier With the Altera LogicLock Flow	10-43
The Xilinx MultiPoint Synthesis Flow	10-48
Using Synplify Pro With Xilinx MultiPoint Synthesis	10-48
Using the Xilinx Modular Flow	10-54
Overview of Modular Flow Design Stages	10-54
Initial Design Budgeting	10-55
Active Implementation	10-58

Final Assembly	10-63
Design Files and Area Design Planning	10-65
Integrating with Third-Party Software	10-70
Resynthesizing with QuickLogic SpDE Information	10-70
Synopsys DesignWare Component Support	10-71
Working with the Identify RTL Debugger	10-72

Chapter 11: Synplify Premier Design Flow

Synplify Premier Physical Synthesis Flows	11-2
Device Support for the Physical Synthesis Flows	11-3
Graph-based Physical Synthesis	11-4
Graph-based Physical Synthesis Design Flow	11-4
Graph-based Physical Synthesis Tasks	11-5
Graph-based Synthesis Flow Specifications	11-6
Graph-based Physical Synthesis with a Design Plan Flow	11-7
Design Plan-based Physical	
Synthesis Flow	11-8
Design Planner Tasks	11-9
Running Physical Synthesis	11-9
Create the Project File	11-10
Set Implementation Options	11-11
Run Place-and-Route	11-18
Synthesize the Design	11-21
Analyze Results	11-22
Running Multiple Implementations	11-24

The FPGA Synthesis Tools

This section briefly discusses the following topics:

- About the Synplify and Synplify Pro Software, next
- About the Synplify Premier Software, on page 1-3
- Synplicity FPGA Tool Features, on page 1-4
- Synplicity Product Family, on page 1-8

About the Synplify and Synplify Pro Software

Synplify® and Synplify Pro® are logic synthesis tools for FPGAs (Field Programmable Gate Arrays) and Complex PLDs (Programmable Logic Devices), developed by Synplicity® of Sunnyvale, California. For input, the software uses high-level designs written in Verilog and VHDL hardware description languages (HDLs). Using proprietary Behavior Extracting Synthesis Technology® (B.E.S.T.)® the tool converts the HDL into small, high-performance, design netlists that are optimized for popular technology vendors. Optionally, the software can write post-synthesis VHDL and Verilog netlists that you can use to verify functionality through simulation.

The Synplify Pro software offers a superset of the Synplify features. For a comparison of the features, see *Synplicity FPGA Tool Features*, on page 1-4. The software has the following built-in features:

- The HDL Analyst® tool, a graphical interface for analysis and crossprobing. This feature is an option with the Synplify software, but is standard with Synplify Pro.
- The Text Editor window, with a language-sensitive editor for writing and editing HDL code.
- The SCOPE® (Synthesis Constraint Optimization Environment®) interface, which uses a spreadsheet-like format to manage the timing constraints and attributes in the design.
- A symbolic FSM Compiler, which performs advanced state machine optimizations.

- Prototyping— The Synplify Premier product supports a complete design and debugging environment featuring the Identify product along with automated HDL code translation for Synopsys® DesignWare® components. Using the Synplify Premier prototyping solution, you can quickly and automatically bring ASIC HDL code into high-capacity FPGAs, and then debug it with real-time feedback from the FPGA. Because the feedback from the debugger goes directly into the RTL code, the integration of logic synthesis lets you correct the code and retest it in the FPGA in record time.

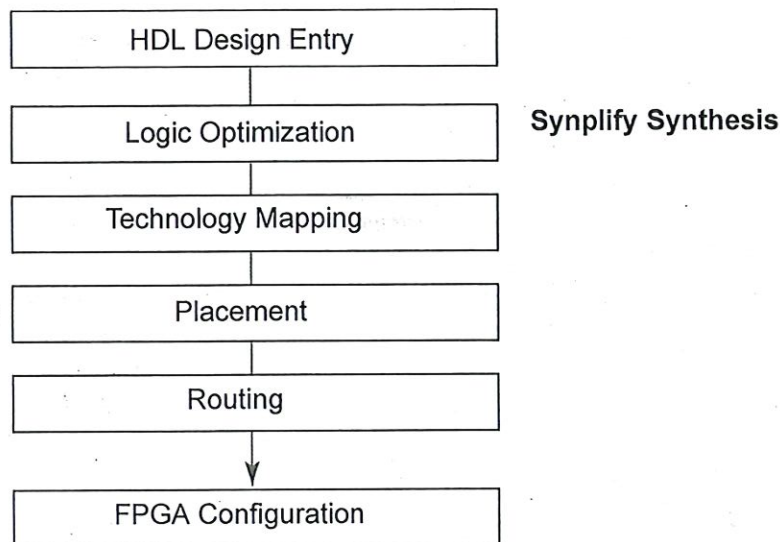
In addition to the features described in *About the Synplify and Synplify Pro Software*, on page 1-2, the Synplify Premier tool features the following:

- Expanded HDL Analyst capabilities including support for physical regions.
- Design Planner - This licensed option for Synplify Premier lets you run interactive physical synthesis using a design plan. You assign physical constraints in Design Planner by interactively dragging and dropping RTL objects into regions of the device. This information along with the normal timing constraints, enables the Synplify Premier tool to estimate timing more accurately and use the estimates for additional optimizations. This produces a more highly optimized circuit in fewer iterations.
- Physical Optimizations - Physical optimization algorithms use physical design characteristics like placement and interconnect delay to affect the actual topology of the circuit. Design plans are used to control the optimizations. The Synplify Premier tool also helps reduce the time required to perform placement and routing, which is especially significant as FPGA designs continue towards the trend of one million plus gates.
- Physical Analysis Tools - These tools include the Physical Analyst and the Island Timing Analyst.

Synplicity FPGA Tool Features

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

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier with Design Planner
Performance				
Behavior Extracting Synthesis Technology® (BEST™)	x	x	x	x
Clearbox Support (selected Altera architectures)		x	x	x
Coregen Support (selected Xilinx architectures)		x	x	x
FSM Compiler	x	x	x	x
FSM Explorer		x	x	x
Gated Clock Conversion		x	x	x
Register Pipelining into Multipliers/ROMs		x	x	x
Register Retiming		x	x	x
Code Analysis				
Timing Analyzer – Point-to-point Report		x	x	x
HDL Analyst® Solution — Graphical Design Analysis	Option	x	x	x
FSM Viewer – View State Transitions		x	x	x
Crossprobing – Cross-tool Analysis		x	x	x
Probe Point Creation – Easy Insertion of Debug Pins		x	x	x
Team Design/Design Environment				
SCOPE® Spreadsheet	x	x	x	x
Advanced Project Management — Workspaces		x	x	x



The following sections describe each step more fully.

HDL Design Entry

The starting point for FPGA design is to specify the logic of the FPGA circuit to be implemented. You can do this by drawing a schematic, writing an HDL description, or specifying Boolean expressions.

For the Synplify flow, design entry is the step where you generate the input for the tool. The input must be Verilog or VHDL descriptions. The software provides you with an environment where you can write or edit HDL descriptions.

Logic Optimization (Compilation)

This is the first stage of synthesis, in which the software restructures the original network into a set of combinational functions. In the Synplify flow, the combinational functions are represented as a Boolean network. At this point in the design process, you modify the initial logic design to optimize the area or speed of the final circuit, or both. The optimization is calculated from the netlist and is independent of the target technology. It includes operations like redundancy removal and common subexpression elimination.

Starting the Software

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

- Getting Started, on page 1-12
- Getting Help, on page 1-13

Getting Started

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

```
synplify  
synplify_pro  
synplify_premier  
synplify_premier_dp
```

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

Synplicity[®] FPGAs Synthesis

*Synplicity[®] Pro, Synplicity[®] Premier, and
Synplicity[®] Premier with Design Planner*

Simply Better Results

Synplicity[®]



Synplicity, Inc.
600 West California Avenue
Sunnyvale, CA 94086
(U.S.) +1 408 215-6000 direct
(U.S.) +1 408 222-0268 fax
www.synplicity.com

User Guide
December 2005

Contents

Chapter 1: Introduction

The FPGA Synthesis Tools	1-2
About the Synplify and Synplify Pro Software	1-2
About the Synplify Premier Software	1-3
Synplify FPGA Tool Features	1-4
Synplify Product Family	1-8
The Generic FPGA Design Flow	1-8
HDL Design Entry	1-9
Logic Optimization (Compilation)	1-9
Technology Mapping	1-10
Placement	1-10
Routing	1-10
FPGA Configuration	1-10
Audience	1-11
Scope of the Document	1-11
Starting the Software	1-12
Getting Started	1-12
Getting Help	1-13
User Interface Overview	1-14

Design Flows	1-16
Logic Synthesis Design Flow	1-16
Prototyping Design Flow	1-17
Physical Synthesis Design Flows	1-17

Chapter 2: Project Setup

Setting Up HDL Source Files	2-2
Creating HDL Source Files	2-2
Checking HDL Source Files	2-4
Editing HDL Source Files with the Built-in Text Editor	2-5

Using an External Text Editor	2-8
Setting Editing Window Preferences	2-9
Setting Up Project Files	2-11
Creating a Project File	2-11
Opening an Existing Project File	2-14
Making Changes to a Project	2-15
Using Mixed Language Source Files	2-16
Setting Project View Display Preferences	2-18
Updating Verilog Include Paths in Older Project Files	2-21
Setting Up Implementations and Workspaces	2-22
Working with Multiple Implementations	2-22
Creating Workspaces	2-24
Using Workspaces	2-25
Archiving Files and Projects	2-26

Chapter 3: Constraints, Attributes, and Options

Setting Implementation Options	3-2
Setting Device Options	3-2
Setting Optimization Options	3-5
Specifying Global Frequency and Constraint Files	3-6
Specifying Result Options	3-9
Specifying Timing Report Output	3-10
Setting Verilog and VHDL Options	3-11
Setting Synplify Premier Netlist Restructuring Optimizations	3-16
Setting Constraints in the SCOPE Window	3-18
Using the SCOPE Window	3-19
Entering and Editing Constraints in the SCOPE Window	3-21
Entering Default Constraints	3-24
Setting Clock and Path Constraints	3-26
Defining Clocks	3-28
Defining Input and Output Constraints	3-33
Defining Multicycle Paths	3-34
Defining From/To/Through for Timing Exceptions	3-35
Defining False Paths	3-38
Specifying Standard I/O Pad Types	3-40
Setting SCOPE Display Preferences	3-41
Translating Xilinx UCF Constraints	3-42
Converting Pin Location Constraint Files in the Synplify Premier Tool	3-43

Using Auto Constraints	3-45
Using Collections	3-47
Comparing Methods for Defining Collections	3-47
Creating and Using Collections (SCOPE Window)	3-48
Creating Collections (Tcl Commands)	3-51
Using the Tcl Find Command to Define Collections	3-53
Using the Expand Tcl Command to Define Collections	3-55
Viewing and Manipulating Collections (Tcl Commands)	3-56
Working with Constraint Files	3-60
When to Use Constraint Files over Source Code	3-60
Tcl Syntax Guidelines for Constraint Files	3-61
Using a Text Editor for Constraint Files	3-62
Generating Constraint Files for Forward Annotation	3-64
Adding Attributes and Directives	3-66
Adding Attributes and Directives in VHDL	3-66
Adding Attributes and Directives in Verilog	3-68
Adding Attributes in the SCOPE Window	3-68
Adding Attributes with the SCOPE Wizard	3-71
Adding Attributes to a Tcl Constraint File	3-73
Adding Attributes From the RTL and Technology Views	3-74

Chapter 4: Result Analysis

Checking Log Results	4-2
Viewing the Log File	4-2
Analyzing Results Using the Log File Reports	4-5
Using the Log Watch Window	4-6
Handling Messages	4-8
Checking Results in the Message Viewer	4-8
Filtering Messages in the Message Viewer	4-10
Filtering Messages from the Command Line	4-13
Handling Warnings	4-14
Automating Message Filtering with a synhooks Script	4-14
Basic Operations in the Schematic Views	4-16
Differentiating Between the Views	4-17
Opening the Views	4-17
Analyzing Your Design Graphically	4-19
Viewing Object Properties	4-20
Selecting Objects in the RTL/Technology Views	4-23
Working with Multisheet Schematics	4-24
Moving Between Views in a Schematic Window	4-26

4-26	Setting Schematic View Preferences
4-28	Managing Windows
4-30	Exploring Design Hierarchy
4-30	Traversing Design Hierarchy with the Hierarchy Browser
4-31	Exploring Object Hierarchy by Pushing/Popping
4-36	Exploring Object Hierarchy of Transparent Instances
4-37	Finding Objects
4-37	Browsing to Find Objects
4-39	Using Find for Hierarchical and Restricted Searches
4-42	Using Wildcards with the Find Command
4-45	Using Find to Search the Output Netlist
4-48	Crossprobing
4-48	Crossprobing Description
4-49	Crossprobing within an RTL/Technology View
4-49	Crossprobing from the RTL/Technology View
4-51	Crossprobing from the Text Editor Window
4-54	Crossprobing from the Tcl Script Window
4-54	Crossprobing from the FSM Viewer
4-56	Analyzing With the HDL Analyst Tool
4-56	Viewing Design Hierarchy and Context
4-60	Filtering Schematics
4-62	Expanding Pin and Net Logic
4-66	Expanding and Viewing Connections
4-67	Flattening Schematic Hierarchy
4-72	Minimizing Memory Usage While Analyzing Designs
4-73	Analyzing Timing
4-73	Analyzing Clock Trees in the RTL View
4-74	Viewing Critical Paths
4-76	Analyzing Paths with the Timing Analyst
4-79	Analyzing Paths with the Synplify Premier Timing Analyst
4-81	Handling Negative Slack
4-83	The Island Timing Report
4-83	Generating the Island Timing Report
4-84	Automatic Island Timing Report
4-85	Defining the Group Range and Global Range
4-86	Interactive Island Timing Analyst
4-87	Viewing the Island Timing Report

4-88	The Island Timing Analyst
4-89	Islands/Paths Control Panel
4-92	Islands/Paths Summary View
4-93	Islands/Paths Summary Management
4-95	Islands/Paths Details View
4-98	Island Timing Report Critical Paths
4-98	Assigning Critical Paths to a Region

Chapter 5: Physical Analyst

5-2	Synplify Premier Physical Analyst Tool
5-4	Opening the Physical Analyst View
5-5	Using the Physical Analyst Control Panel
5-6	Setting Object Controls
5-6	Setting Visibility Controls
5-8	Using the Physical Analyst Device View
5-10	Setting Object Display Options
5-10	Displaying Instances
5-10	Displaying Signal Pins
5-11	Displaying Signal Flow for Selected Nets
5-12	Routing Nets to Display
5-14	Selecting Objects
5-15	Selecting Multiple Nets
5-16	Transcribing Object Selections
5-17	Viewing Object Information
5-17	Viewing Properties
5-20	Using Tool Tips
5-20	Using Mouse Strokes
5-21	Using Keyboard Shortcuts
5-22	Zooming in the Physical Analyst
5-23	Finding Objects
5-23	Finding Objects with the Find Command
5-27	Finding Object Locations
5-29	Using Markers
5-31	Changing Color Schemes
5-31	Configuring Enhanced Instance Display
5-34	Crossprobing in Physical Analyst

6-44	Retiming
6-44	Controlling Retiming
6-46	Retiming Example
6-48	Retiming Report
6-48	How Retiming Works
6-50	How Retiming Works With Synplify Premier Regions
6-50	Inserting Probes
6-51	Specifying Probes in the Source Code
6-52	Adding Probe Attributes Interactively
6-54	Infering RAMs
6-54	Inference vs. Instantiation
6-55	Coding RAMs for Inference
6-59	Specifying RAM Implementation Styles
6-61	Implementing Altera RAMs Automatically
6-64	Implementing Xilinx RAMs Automatically
6-67	Implementing Altera RAMs: FLEX and APEX
6-69	Implementing Altera RAMs: Stratix Multi-Port RAMs
6-70	Infering Xilinx Block RAMs Using Registered Addresses
6-73	Infering Xilinx Block RAMs Using Registered Output
6-78	Setting Xilinx RAM Initialization Values
6-79	Mapping Xilinx ROM to Block RAM
6-80	Infering Shift Registers
6-82	Shift Register Examples
6-86	Forward Annotation of Initial Values
6-87	Working with LPMs
6-88	Instantiating LPMs as Black Boxes (Altera)
6-92	Instantiating LPMs as Black Boxes (Cypress)
6-94	Instantiating LPMs Using VHDL Prepared Components
6-97	Instantiating LPMs Using a Verilog Library (Altera)
6-99	Working with Gated Clocks
6-99	The Synplify Approach to Gated Clocks
6-101	Synthesizing a Gated Clock Design
6-103	Prerequisites for Gated Clock Conversion
6-105	Gated Clock Conversion Report
6-106	Fix Gated Clock Error Messages
6-108	Gated Clocks for Black Boxes
6-110	Restrictions to Using Fix Gated Clocks
6-111	Generated-Clock Optimization

Chapter 7: Design Planning and Optimizations

Using the Design Planner	7-2
Creating a Design Plan	7-2
Cutting, Copying, and Pasting in the Design Planner	7-5
Pin Assignments	7-6
Methods for Specifying Pin Assignments	7-6
Specifying Pins Using the Design Plan Editor	7-7
Implementing Pin Assignments	7-11
Storing Temporary Pin Assignments	7-14
Displaying Rats Nesting	7-15
Pin Assignment Statistics	7-16
Assigning Clock Pins	7-17
Working with Regions	7-19
Viewing Intellectual Property (IP) Core Areas	7-19
Placing Regions	7-20
Moving and Sizing Regions	7-21
Replicating Logic Manually	7-23
Assigning Register to Pin-Locked I/O Paths to Regions	7-24
Checking Synplify Premier Utilization	7-25
Device Utilization	7-25
Region Utilization	7-25
Using Process-Level Hierarchy	7-25
Bit Slicing	7-26
About Bit Slicing	7-26
Using Bit Slicing	7-26
Bit Slice Examples	7-29
Bit Slicing Guidelines	7-32
Zippering	7-33
Using Zippering	7-33
Analyzing a Design for Zippering	7-36
Zippering Example	7-37
Zippering Guidelines	7-41

5-42	Analyzing Netlist with the Physical Analyst
5-48	Expanding and Viewing Connections
5-51	Analyzing Timing with the Physical Analyst
5-51	Viewing Critical Paths
5-54	Tracing Critical Paths Forward and Backwards

Chapter 6: Design Optimization

6-2	Design Guidelines
6-2	General Optimization Tips
6-3	Area Optimization Tips
6-4	Timing Optimization Settings
6-5	Optimizing Results
6-5	Sharing Resources
6-7	Setting Fanout Limits
6-8	Controlling Buffering and Replication
6-10	Controlling Hierarchy Flattening
6-10	Preserving Objects from Optimization
6-12	Preserving Hierarchy
6-13	Defining State Machines for Synthesis
6-13	Defining State Machines in Verilog
6-14	Defining State Machines in VHDL
6-15	Specifying FSMs with Attributes and Directives
6-17	Using the Symbolic FSM Compiler
6-17	Choosing When to Use the FSM Compiler
6-18	Running the FSM Compiler on the Whole Design
6-20	Running the FSM Compiler on Individual FSMs
6-22	Using FSM Explorer
6-22	Deciding When to Use the FSM Explorer
6-23	Running the FSM Explorer
6-25	Using the FSM Viewer
6-30	Defining Black Boxes for Synthesis
6-30	Instantiating Black Boxes and I/Os in Verilog
6-32	Instantiating Black Boxes and I/Os in VHDL
6-34	Adding Black Box Timing Constraints
6-38	Adding Other Black Box Attributes
6-40	Pipelining
6-40	Prerequisites for Pipelining
6-41	Pipelining the Design

Chapter 8: Vendor-Specific Optimizations

8-2	Passing Information to the P&R Tools
8-2	Specifying Pin Locations
8-3	Specifying Locations for Actel Bus Ports
8-3	Specifying Macro and Register Placement
8-4	Passing Technology Properties
8-5	Specifying Padtype and Port Information
8-6	Generating Vendor-Specific Output
8-6	Targeting Output to Your Vendor
8-7	Customizing Netlist Formats
8-8	Working with Actel Designs
8-8	Using Predefined Actel Black Boxes
8-9	Using ACTGen Macros
8-10	Working with Radhard Designs
8-11	Working with Altera Designs
8-12	APEX Design Tips
8-12	FLEX Design Tips
8-12	Determining ROM Implementation
8-14	Working with Altera EABs and ESBs
8-15	Working with Altera PLLs
8-16	Implementing Megafuncions with Clearbox
8-18	Packing I/O Cell Registers
8-20	Using LPMs in Simulation Flows
8-22	Working with Quartus II
8-23	Working with Lattice Designs
8-23	Instantiating Lattice Macros
8-24	Using Lattice GSR Resources
8-25	Inferring Carry Chains in Lattice XPLD Devices
8-25	Controlling I/O Insertion in Lattice Designs
8-26	Forward-Annotating Lattice ORCA Constraints
8-28	Working with Xilinx Designs
8-28	Designing for Xilinx Architectures
8-29	Instantiating CoreGen Cores
8-30	Instantiating Virtex PCI Cores
8-33	Packing Registers for I/Os
8-35	Controlling Placement with RLOCs
8-36	Using Clock Buffers in Virtex Designs
8-38	Instantiating Special I/O Standard Buffers for Virtex
8-39	Reoptimizing With EDIF Files
8-39	Working with Xilinx Place-and-Route Software

Chapter 9: Design Planning for Vendors

Design Planning with Altera Devices	9-2
Stratix and Cyclone Devices	9-2
Displaying Stratix Devices	9-2
Creating Regions for Stratix Devices	9-4
Design Planning with Xilinx Designs	9-8
Displaying Xilinx Device Resources	9-8
Creating Regions for Xilinx Designs	9-12
Handling Xilinx Critical Paths (Design Planner)	9-14
Splitting a Critical Path into Multiple Regions	9-14
Creating Smaller Regions for Long Critical Paths	9-15
Handling Critical Paths with High Fanout Nets	9-15
Handling Critical Paths with Cascading Cells or Carry Chain Logic	9-17
Handling Critical Paths with Bit Slicing	9-19
Handling Critical Paths with Pipelining	9-20
Handling Designs with Multiple Critical Paths	9-20
Handling Critical Paths with Large Multiplexers	9-21
Handling Xilinx Black Boxes (Design Planner)	9-22
Design Planning Xilinx Black Boxes	9-22
Handling Xilinx Block RAMs (Design Planner)	9-24
Creating Block RAM Regions	9-25
Assigning to Block RAM Regions	9-27
Handling Block Multipliers (Design Planner)	9-30
Block Multiplier Support	9-30
Creating Block Mult Regions	9-30
Assigning to Block Mult Regions	9-31
Region Utilization	9-31
Handling DSP Blocks (Design Planner)	9-32
Handling Xilinx IPs (Design Planner)	9-35
Intrusive IP Flow	9-35
Macro IP Flow	9-37
Creating IP Region Constraints	9-38

Chapter 10: Design Flows and Process Optimization

10-2	Using Batch Mode
10-2	Running Batch Mode on a Project File
10-3	Running Batch Mode with a Tcl Script
10-4	Working with Tcl Scripts and Commands
10-4	Crossprobing from the Tcl Script Window
10-4	Using Tcl Commands and Scripts
10-5	Generating a Job Script
10-5	Creating a Tcl Synthesis Script
10-7	Using Tcl Variables to Try Different Clock Frequencies
10-8	Using Tcl Variables to Try Several Target Technologies
10-9	Running Bottom-up Synthesis with a Script
10-10	Automating Flows with synhooks.tcl
10-13	The VIF Formal Verification Flow
10-13	Overview of the VIF Flow
10-14	Generating a VIF File
10-16	Using a Tcl Script for VIF Conversion
10-18	Handling Equivalency Check Failures
10-19	Protected Flow Support
10-20	Using IP in a Design
10-21	Running Place-and-Route After Synthesis
10-21	Creating and Running P&R Projects
10-23	Specifying Xilinx Place-and-Route Options
10-25	Backannotating Place-and-Route Data
10-26	Analyzing Physical Synthesis (Synplify Premier)
10-28	MultiPoint Synthesis
10-28	Traditional Bottom-up Design and MultiPoint Synthesis
10-29	The Synplify Pro MultiPoint Synthesis Flow
10-39	The Altera LogicLock Flow
10-39	Using Synplify Pro With the Altera LogicLock Flow
10-43	Using Synplify Premier With the Altera LogicLock Flow
10-48	The Xilinx MultiPoint Synthesis Flow
10-48	Using Synplify Pro With Xilinx MultiPoint Synthesis
10-54	Using the Xilinx Modular Flow
10-54	Overview of Modular Flow Design Stages
10-55	Initial Design Budgeting
10-58	Active Implementation

Final Assembly	10-63
Design Files and Area Design Planning	10-65
Integrating with Third-Party Software	10-70
Resynthesizing with QuickLogic SpDE Information	10-70
Synopsys DesignWare Component Support	10-71
Working with the Identify RTL Debugger	10-72

Chapter 11: Synplify Premier Design Flow

Synplify Premier Physical Synthesis Flows	11-2
Device Support for the Physical Synthesis Flows	11-3
Graph-based Physical Synthesis	11-4
Graph-based Physical Synthesis Design Flow	11-4
Graph-based Physical Synthesis Tasks	11-5
Graph-based Synthesis Flow Specifications	11-6
Graph-based Physical Synthesis with a Design Plan Flow	11-7
Design Plan-based Physical Synthesis Flow	11-8
Design Planner Tasks	11-9
Running Physical Synthesis	11-9
Create the Project File	11-10
Set Implementation Options	11-11
Run Place-and-Route	11-18
Synthesize the Design	11-21
Analyze Results	11-22
Running Multiple Implementations	11-24

The FPGA Synthesis Tools

This section briefly discusses the following topics:

- About the Synplify and Synplify Pro Software, next
- About the Synplify Premier Software, on page 1-3
- Synplify FPGA Tool Features, on page 1-4
- Synplify Product Family, on page 1-8

About the Synplify and Synplify Pro Software

Synplify® and Synplify Pro® are logic synthesis tools for FPGAs (Field Programmable Gate Arrays) and Complex PLDs (Programmable Logic Devices), developed by Synplify® of Sunnyvale, California. For input, the software uses high-level designs written in Verilog and VHDL hardware description languages (HDLs). Using proprietary Behavior Extracting Synthesis Technology® (B.E.S.T.)® the tool converts the HDL into small, high-performance, design netlists that are optimized for popular technology vendors. Optionally, the software can write post-synthesis VHDL and Verilog netlists that you can use to verify functionality through simulation.

The Synplify Pro software offers a superset of the Synplify features. For a comparison of the features, see *Synplify FPGA Tool Features*, on page 1-4. The software has the following built-in features:

- The HDL Analyst® tool, a graphical interface for analysis and crossprobing. This feature is an option with the Synplify software, but is standard with Synplify Pro.
- The Text Editor window, with a language-sensitive editor for writing and editing HDL code.
- The SCOPE® (Synthesis Constraint Optimization Environment®) interface, which uses a spreadsheet-like format to manage the timing constraints and attributes in the design.
- A symbolic FSM Compiler, which performs advanced state machine optimizations.

- Prototyping—The Synplify Premier product supports a complete design and debugging environment featuring the Identify product along with automated HDL code translation for Synopsys® DesignWare® components. Using the Synplify Premier prototyping solution, you can quickly and automatically bring ASIC HDL code into high-capacity FPGAs, and then debug it with real-time feedback from the FPGA. Because the feedback from the debugger goes directly into the RTL code, the integration of logic synthesis lets you correct the code and retest it in the FPGA in record time.

In addition to the features described in *About the Synplify and Synplify Pro Software*, on page 1-2, the Synplify Premier tool features the following:

- Expanded HDL Analyst capabilities including support for physical regions.
- Design Planner - This licensed option for Synplify Premier lets you run interactive physical synthesis using a design plan. You assign physical constraints in Design Planner by interactively dragging and dropping RTL objects into regions of the device. This information along with the normal timing constraints, enables the Synplify Premier tool to estimate timing more accurately and use the estimates for additional optimizations. This produces a more highly optimized circuit in fewer iterations.
- Physical Optimizations - Physical optimization algorithms use physical design characteristics like placement and interconnect delay to affect the actual topology of the circuit. Design plans are used to control the optimizations. The Synplify Premier tool also helps reduce the time required to perform placement and routing, which is especially significant as FPGA designs continue towards the trend of one million plus gates.
- Physical Analysis Tools - These tools include the Physical Analyst and the Island Timing Analyst.

Synplify FPGA Tool Features

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

	Performance			
	Synplify	Synplify Pro	Synplify Premier	Synplify Premier with Design Planner
Behavior Extracting Synthesis Technology® (BEST™)	x	x	x	x
Clearbox Support (selected Altera architectures)		x	x	x
Coregen Support (selected Xilinx architectures)		x	x	x
FSM Compiler	x	x	x	x
FSM Explorer		x	x	x
Gated Clock Conversion		x	x	x
Register Pipelining into Multipliers/ROMs		x	x	x
Register Retiming		x	x	x
Code Analysis				
Timing Analyzer – Point-to-point Report		x	x	x
HDL Analyst® Solution – Graphical Design Analysis	Option	x	x	x
FSM Viewer – View State Transitions		x	x	x
Crossprobing – Cross-tool Analysis		x	x	x
Probe Point Creation – Easy Insertion of Debug Pins		x	x	x
Team Design/Design Environment				
SCOPE® Spreadsheet	x	x	x	x
Advanced Project Management – Workspaces		x	x	x

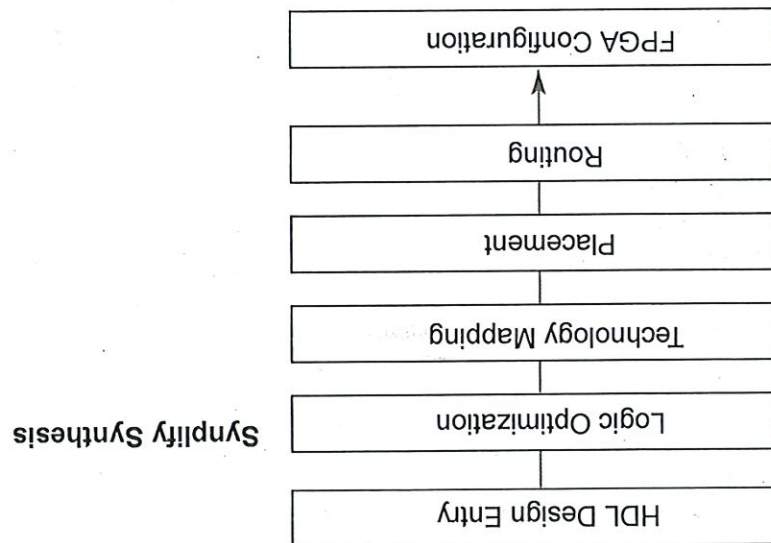
This is the first stage of synthesis, in which the software restructures the original network into a set of combinational functions. In the Synplify flow, the combinational functions are represented as a Boolean network. At this point in the design process, you modify the initial logic design to optimize the area or speed of the final circuit, or both. The optimization is calculated from the netlist and is independent of the target technology. It includes operations like redundancy removal and common subexpression elimination.

Logic Optimization (Compilation)

The starting point for FPGA design is to specify the logic of the FPGA circuit to be implemented. You can do this by drawing a schematic, writing an HDL description, or specifying Boolean expressions. For the Synplify flow, design entry is the step where you generate the input for the tool. The input must be Verilog or VHDL descriptions. The software provides you with an environment where you can write or edit HDL descriptions.

HDL Design Entry

The following sections describe each step more fully.



Starting the Software

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

- Getting Started, on page 1-12
- Getting Help, on page 1-13

Getting Started

1. If you have not already done so, install the Synplify software according to the installation instructions.

2. Start the software.

- If you are working on a Windows platform, select Programs->Synplify->product version from the Start button.
- If you are working on a UNIX platform, type the appropriate command at the command line:

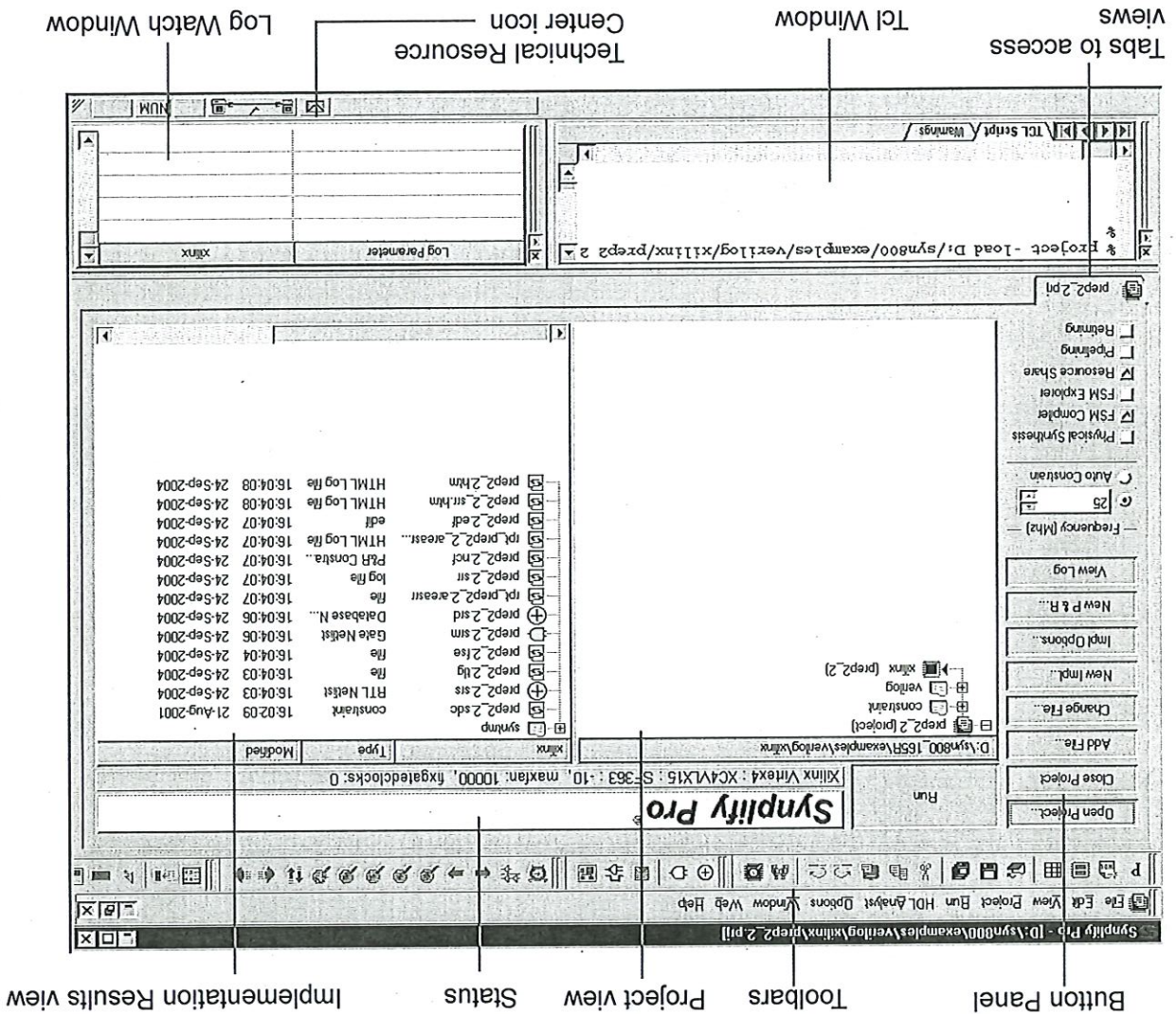
```
synplify  
synplify_pro  
synplify_premier  
synplify_premier_dp
```

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

User Interface Overview

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

Synplify Pro and Synplify Premier Standard Interface



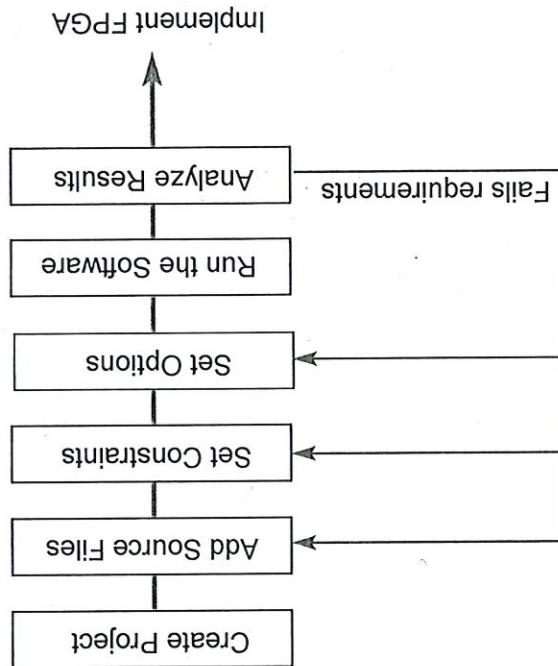
Design Flows

This section provides an overview of the following flows:

- Logic Synthesis Design Flow, on page 1-16
- Prototyping Design Flow, on page 1-17
- Physical Synthesis Design Flows, on page 1-17

Logic Synthesis Design Flow

The following figure shows a logic synthesis design flow using Synplify, Synplify Pro, and Synplify Premier synthesis software. For a design flow with step-by-step instructions based on specific design data, download the tutorial from the Synplify website:
<http://trc.synplify.com/tutorials/index.html>.



CHAPTER 2

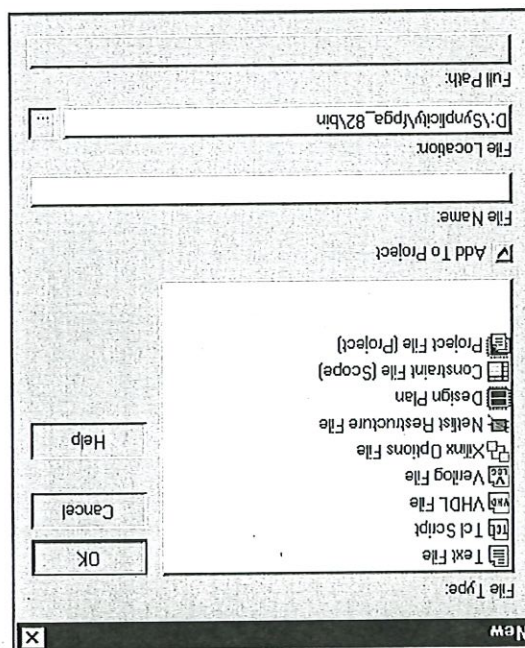
Project Setup

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

- Setting Up HDL Source Files, on page 2-2
- Setting Up Project Files, on page 2-11
- Setting Up Implementations and Workspaces, on page 2-22
- Archiving Files and Projects, on page 2-26

- Type a name and location for the file and Click OK.
- A blank editing window opens with line numbers on the left. You can name it now by pressing Ctrl-s and naming the file.

2. Type the source information in the window, or cut and paste it. See *Editing HDL Source Files with the Built-in Text Editor*, on page 2-5 for more information on working in the Editing window.
 - For the best synthesis results, check the *Reference Manual* and ensure that you are using the available constructs and vendor-specific attributes and directives effectively.
 3. Save the file by selecting File->Save or the Save icon (floppy disk). Use the correct extension for the type of file you created (.v or .vhdl).
- Once you have created a source file, you can check that you have the right syntax, as described in *Checking HDL Source Files*, on page 2-4.



6. Repeat steps 4 and 5 until all syntax and synthesis errors are corrected. Messages can be categorized as errors, warnings, or notes. Review all messages and resolve any errors. Warnings are less serious than errors, but you must read through and understand them even if you do not resolve all of them. Notes are informative and do not need to be resolved.

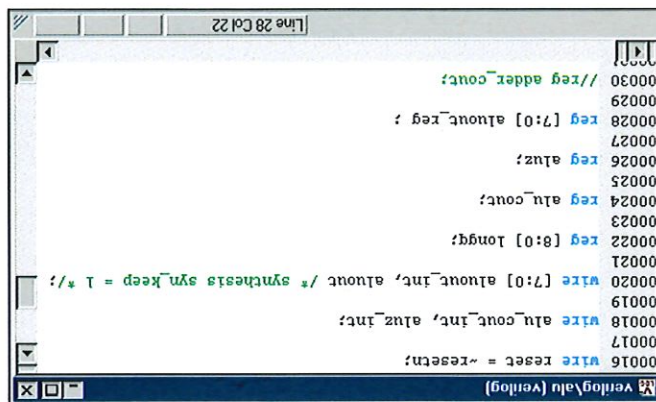
Editing HDL Source Files with the Built-in Text Editor

The built-in text editor makes it easy to create your HDL source code, view it, or edit it when you need to fix errors. If you want to use an external text editor, see [Using an External Text Editor, on page 2-8](#).

1. Do one of the following to open a source file for viewing or editing:

- To automatically open the first file in the list with errors, press F5.
- To open a specific file, double-click the file in the Project window or use File->Open (Ctrl-O) and specify the source file.

The Text Editor window opens and displays the source file. Lines are numbered. Keywords are in blue, and comments in green. String values are in red. If you want to change these colors, see [Setting Editing Window Preferences, on page 2-9](#).



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

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

- Open the HTML log file (not available with the Synplify product) and click on the 5-character error code.
- In the Tcl window (not available with the Synplify product), click the Messages tab and click on the 5-character error code in the ID column.
- 7. To crossprobe from the source code window to other views, open the view and select the piece of code. See *Crossprobing from the Text Editor Window*, on page 4-51 for details.
- 8. When you have fixed all the errors, select File->Save or click the Save icon to save the file.

Using an External Text Editor

You can use an external text editor like vi or emacs instead of the built-in text editor. Do the following to enable an external text editor. For information about using the built-in text editor, see *Editing HDL Source Files with the Built-in Text Editor*, on page 2-5.

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

Setting Editing Window Preferences

You can customize the fonts and colors used by the internal editor in the Text Editing window.

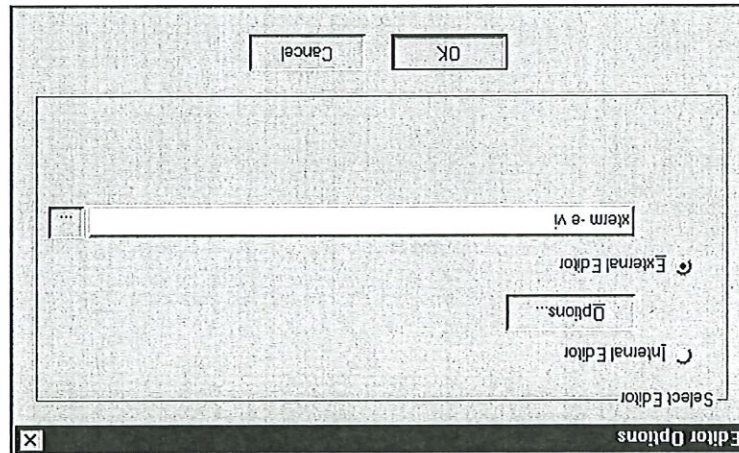
1. Select Options->Editor Options, and select Internal Editor. Click Options.
2. Select the kind of file for which you want to set the preferences.
3. This table shows you how to set some common syntax options from the Editor Options form:

The Text Editing window can be used to set preferences for source files, log files, Tcl files, constraint files, or other default files. The Editor Options form opens.

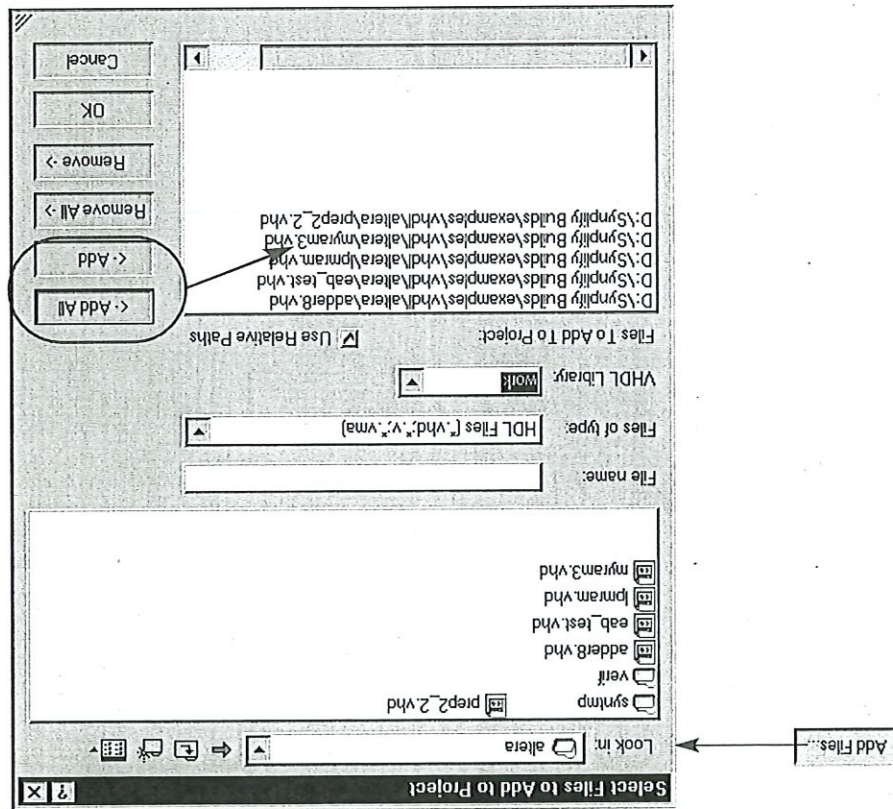
3. Click OK.

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

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



- To add all the files in the directory at once, click the Add All button on the right side of the form. To add files individually, click on the file in the list and then click the Add button, or double-click the file name. You can add all the files in the directory and then remove the ones you do not need with the Remove button.
- If you are adding VHDL files, select the appropriate library from the the VHDL Library popup menu. The library you select is applied to all VHDL files when you click OK in the dialog box.
- Your project window displays a new project file. If you click on the plus sign next to the project and expand it, you see the following:
 - A folder (two folders for mixed language designs) with the source files. If your files are not in a folder under the project directory, you can set this preference by selecting Options->Project View Options and checking the View project files in folders box. This separates one kind of file from another in the Project view by putting them in separate folders.



Setting Up Project Files

For a specific example on setting up a project file, refer to the Synplify and Synplify Pro tutorial. This section describes the following:

- Creating a Project File, next
- Opening an Existing Project File, on page 2-14
- Making Changes to a Project, on page 2-15
- Using Mixed Language Source Files, on page 2-16
- Setting Project View Display Preferences, on page 2-18
- Updating Verilog Include Paths in Older Project Files, on page 2-21

Creating a Project File

You must set up a project file for each project. A project contains the data needed for a particular design: the list of source files, the synthesis results file, and your device option settings. The following procedure shows you how to set up a project file using individual commands.

1. Start by selecting one of the following: File->Build Project, File->Open Project, or the P icon. Click New Project.
The Project window shows a new project. Click the Add File button, press F4, or select the Project->Add Source File command. The Select Files to Add to Project dialog box opens.

2. Add the source files to the project.

— Make sure the Look in field at the top of the form points to the right directory. The files are listed in the box. If you do not see the files, check that the Files of Type field is set to display the correct file type. If you have mixed input files, follow the procedure described in *Using Mixed Language Source Files*, on page 2-16.

Open Project Command	File->Open Command
Select File->Open Project, click the Open Project button on the left side of the Project window (Synplify Pro and Synplify Premier only), or click the P icon.	Select File->Open. Specify the correct directory in the Look In: field.
To open a recent project, double-click it from the list of recent projects.	Set File of Type to Project Files (*.prj). The box lists the project files.
Otherwise, click the Existing Project button to open the Open dialog box and select the project.	Double-click on the project you want to open.

The project opens in the Project window.

Making Changes to a Project

Typically, you might have to add, delete, or replace files.

1. To add source or constraint files to a project, select the Add Files button or Project->Add Source File to open the Select Files to Add to Project dialog box. See *Creating a Project File*, on page 2-11 for details.
2. To delete a file from a project, click the file in the Project window, and press the Delete key.
3. To replace a file in a project,
 - Select the file you want to change in the Project window.
 - Click the Change File button, or select Project->Change File.
 - In the Source File dialog box that opens, set Look In to the directory where the new file is located. The new file must be of the same type as the file you want to replace.
 - If you do not see your file listed, select the type of file you need from the Files of Type field.
 - Double-click the file. The new file replaces the old one in the project list.

4. To specify how project files are saved in the project, right click on a file in the Project view and select File Options. Set the Save File option to either Relative to Project or Absolute Path.
5. To check the time stamp on a file, right click on a file in the Project view and select File Options. Check the time that the file was last modified. Click OK.

Using Mixed Language Source Files

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

1. Remember these restrictions and set up the mixed language design files accordingly:

- You can not use defparams across languages.
- Verilog does not support unconstrained VHDL ports

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

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

3. When you open a project or create a new one, add the Verilog and VHDL files as follows:

- Select the Project->Add Source File command or click the Add File button.
- On the form, set Files of Type to HDL Files (*.vhd, *.vhdl, *.v).
- Select the Verilog and VHDL files you want and add them to your project. Click OK. For details about adding files to a project, see *Making Changes to a Project*, on page 2-15.

Setting Implementation Options

You can set global options for your synthesis implementations, some of them technology-specific. This section describes how to set global options like device, optimization, and file options with the Implementation Options command. For information about setting constraints for the implementation, see *Setting Constraints in the SCOPE Window*, on page 3-18. For information about overriding global settings with individual attributes or directives, see *Adding Attributes and Directives*, on page 3-66.

This section discusses the following topics:

- Setting Device Options, next
- Setting Optimization Options, on page 3-5
- Specifying Global Frequency and Constraint Files, on page 3-6
- Specifying Result Options, on page 3-9
- Specifying Timing Report Output, on page 3-10
- Setting Verilog and VHDL Options, on page 3-11
- Setting Synplify Premier Netlist Restructuring Optimizations, on page 3-16

Setting Device Options

Device options are part of the global options you can set for the synthesis run. They include the part selection (technology, part and speed grade) and implementation options (I/O insertion and fanouts). The options and the implementation of these options can vary from technology to technology, so check the vendor chapters of the *Reference Manual* for information about your vendor options.

1. Open the Options for Implementation form by clicking the Impl Options button or selecting Project->Implementation Options, and click the Device tab at the top if it is not already selected.

2. Select the technology, part, package, and speed. Available options vary, depending on the technology you choose. Also, the Synplify Premier

software does not support as many technologies as the Synplify and Synplify Pro tools do.

The screenshot shows the 'Device' dialog box in Synplify Pro. It has two tabs: 'Xilinx Virtex4' and 'Actel PA'. The 'Xilinx Virtex4' tab is selected, displaying the following fields: 'Part' with the value 'XC4VLX15', 'Speed' with the value '-10', and 'Package' with the value 'SF363'. The 'Actel PA' tab is also visible, showing 'Part' as 'APA075' and 'Speed' as 'Std'.

3. Set the device mapping options. The options vary, depending on the technology you choose.

- If you are unsure of what an option means, click on the option to see a description in the box below. For full descriptions of the options, click F1 or refer to the appropriate vendor chapter in the *Reference Manual*.

- To set an option, type in the value or check the box to enable it.

For more information about setting fanout limits, pipelining, and retiming, see *Setting Fanout Limits*, on page 6-7, *Pipelining*, on page 6-40, and *Retiming*, on page 6-44, respectively. For details about other vendor-specific options, refer to the appropriate vendor chapter and technology family in the *Reference Manual*. Note that the Synplify tool does not support all these optimization options.

The screenshot shows the 'Device Mapping Options' dialog box. It contains a table of options with checkboxes and values. The 'Fanout Guide' option is checked, and its value is 10000. The 'Disable I/O Insertion' option is checked. The 'Update Compile Point Timing Data' option is checked. The 'Verification Mode' option is checked. There are also links to click on an option for a description.

Option	Value
Fanout Guide	10000
Disable I/O Insertion	☒
Pipelining	☒
Update Compile Point Timing Data	☒
Verification Mode	☒

4. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 for a list of choices). Click OK.

5. Click the Run button to synthesize the design.

The software compiles and maps the design using the options you set.

6. To set device options with a script, use the set_option Tcl command.

The following table contains an alphabetical list of the device options on the form mapped to the equivalent Tcl commands. Because the options are technology-based, all the options will not apply to your design. All commands begin with set_option, followed by the syntax in the column as shown. Check the *Reference Manual* for the most comprehensive list of options for your vendor.

The following table shows typical device options.

Option	Tcl Command (set_option...)
Area delay percent (Altera, Lattice, Xilinx)	-area_delay_percent net_percentage
Cliquing (Altera)	-cliquing {true false}
Disable I/O insertion	-disable_io_insertion {true false}
Fan-in limit (Altera, Lattice)	-fanin_limit max_fanin
Fanout guide (Actel, Atmel, Xilinx)	-fanout_guide fanout_value
Fanout limit	-fanout_limit limit
Fanout limit (hard) (Actel)	-maxfan_hard {true false}
Force GSR usage (Lattice, Xilinx)	-force_gsr {true false}
Map logic (Altera, Lattice, Xilinx)	-map_logic {true false}
Maximum terms/macroc cell (Lattice, Xilinx)	-max_terms_per_macrocell max_terms
Package	-package pkg_name
Part	-part part_name
Soft buffers (Altera)	-soft_buffers {true false}

Option	Tcl Command (set_option...)
Speed	-speed_grade speed_grade
Technology	-technology keyword

Setting Optimization Options

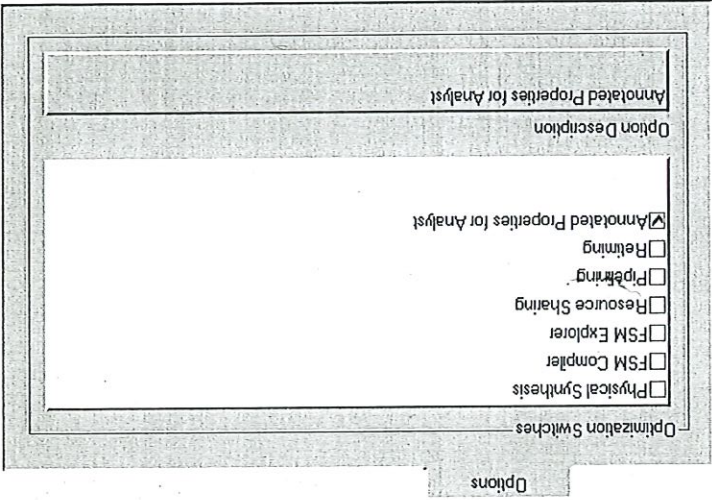
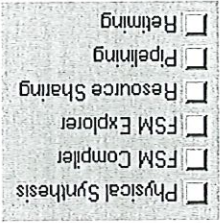
Optimization options are part of the global options you can set for the implementation. This section tells you how to set options like frequency and global optimization options like resource sharing. You can also set some of these options with the appropriate buttons on the UI.

1. Open the Options for Implementation form by clicking the Impl Options button or selecting Project->Implementation Options, and click the Options tab at the top.

2. Click the optimization options you want, either on the form or in the Project view. Your choices vary, depending on the technology. If an option is not available for your technology, it is grayed out. Setting the option in one place automatically updates it in the other. The Synplify software does not support all the options shown below.

Optimization Options

Project View Implementation Options->Options



For details about using these optimizations refer to the following sections:

FSM Compiler	<i>Using the Symbolic FSM Compiler</i> , on page 6-17
FSM Explorer	<i>Using FSM Explorer</i> , on page 6-22
Resource Sharing	<i>Sharing Resources</i> , on page 6-5
Pipelining	<i>Pipelining</i> , on page 6-40
Retiming	<i>Retiming</i> , on page 6-44
Annotated Properties for Analyst	Annotates the design with generic non-timing instance properties (.sap) and timing properties (.tap). These properties are then viewable in the RTL View and Design Planner, as well as used to create collections using TCL. Find. See <i>Object Properties</i> , on page 5-54 and <i>Annotated Timing Information</i> , on page 6-34 for more information.

The equivalent Tcl set_option command options are -frequency, -frequency auto, -resource_sharing, -use_fsm_explorer, -pipe, -retiming and -symbolic_fsm_compiler.

3. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 for a list of choices). Click OK.

4. Click the Run button to run synthesis.

The software compiles and maps the design using the options you set.

Specifying Global Frequency and Constraint Files

This procedure tells you how to set the global frequency and specify the constraint files for the implementation.

1. To set a global frequency, do one of the following:

- Type a global frequency in the Project view.
- Open the Options for Implementation form by clicking the Impl Options button or selecting Project>Implementation Options, and click the Constraints tab.

The equivalent Tcl set_option commands is -frequency *frequency_value*.

You can override the global frequency with local constraints, as described in *Setting Constraints in the SCOPE Window*, on page 3-18. In the Synplify Pro tool, you can automatically generate clock constraints for your design instead of setting a global frequency. See *Using Auto Constraints*, on page 3-45 for details.

Global Frequency and Constraints

Project View

Frequency (Mhz) — — ☐ Auto Constrain

Implementation Options->Constraints

Constraints

Frequency (Mhz) ☐ Maximize (Optimize to obtain maximum frequency)

☐ Use clock period for unconstrained IO

Constraint Files

Check the "Set" column for files that apply to this implementation

Set	File	Module
☐		
☐		
☐		
☐		
☐		

2. To specify constraint files for an implementation, do one of the following:
 - Select Project->Implementation Options->Constraints. Check the constraint (.sdc) files you want to use in the project.
 - With the implementation you want to use selected, click Add File in the Project view, and add the constraint files you need.
- To create constraint files, see *Setting Constraints in the SCOPE Window*, on page 3-18.
3. To remove constraint files from an implementation, do one of the following:
 - Select Project->Implementation Options->Constraints. Click off the checkbox next to the file name.

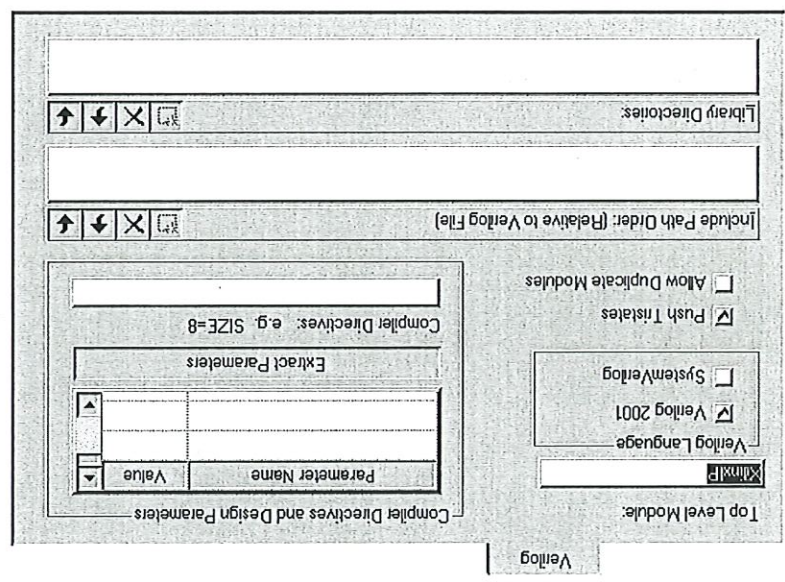
When you synthesize the design, the software compiles and maps the design using the options you set.

Setting Verilog and VHDL Options

When you set up the Verilog and VHDL source files in your project, you can also specify certain compiler options.

Setting Verilog File Options

You set Verilog file options by selecting either Project->Implementation Options->Verilog, or Options->Configure Verilog Compiler. For information about creating always block hierarchy for Synplify Premier, see *Setting Synplify Premier Netlist Restructuring Optimizations*, on page 3-16.



1. Specify the Verilog format to use.

- To set the compiler globally for all the files in the project, select Project->Implementation Options->Verilog. If you are using Verilog 2001, check the *Reference Manual* for supported constructs.
- To specify the Verilog compiler on a per file basis, select the file in the Project view. Right-click and select File Options. Select the appropriate compiler. The default is Verilog 2001.

Setting Constraints in the SCOPE Window

You can use a text editor to create a constraint file as described in *Working with Constraint Files*, on page 3-60, but it is easier to use the SCOPE (Synthesis Constraint Optimization Environment) window, which provides a spreadsheet-like interface for entering constraints. The SCOPE interface is good for editing most constraints, but there are some constraints (like black box constraints) which can only be entered as directives in the source files. If you want to use a text editor to edit a constraint file, close the SCOPE window before editing the file, or you will overwrite results.

This section describes the following:

- Using the SCOPE Window, next
- Entering and Editing Constraints in the SCOPE Window, on page 3-21
- Entering Default Constraints, on page 3-24
- Setting Clock and Path Constraints, on page 3-26
- Defining Clocks, on page 3-28
- Defining Input and Output Constraints, on page 3-33
- Defining Multicycle Paths, on page 3-34
- Defining From/To/Through for Timing Exceptions, on page 3-35
- Defining False Paths, on page 3-38
- Specifying Standard I/O Pad Types, on page 3-40
- Setting SCOPE Display Preferences, on page 3-41
- Translating Xilinx UCF Constraints, on page 3-42
- Converting Pin Location Constraint Files in the Synplify Premier Tool, on page 3-43

You can also use the SCOPE window to do the following:

- Add attributes. See *Adding Attributes in the SCOPE Window*, on page 3-68.
- Create collections and apply constraints to them. See *Creating and Using Collections (SCOPE Window)*, on page 3-48.

Working with Constraint Files

Constraint files are text files that are automatically generated by the SCOPE interface (see *Setting Constraints in the SCOPE Window*, on page 3-18), or which you create manually with a text editor. They contain Tcl commands or attributes that constrain the synthesis run. Alternatively, you can set constraints in the source code, but it is not the preferred method.

This section contains information about

- When to Use Constraint Files over Source Code, next
- Tcl Syntax Guidelines for Constraint Files, on page 3-61
- Using a Text Editor for Constraint Files, on page 3-62
- Generating Constraint Files for Forward Annotation, on page 3-64

When to Use Constraint Files over Source Code

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

However, if you have black box timing constraints like `syn_tco`, `syn_tpd`, and `syn_tsu`, you must enter them as directives in the source code. Unlike attributes, directives can only be added to the source code, not to constraint files. See *Adding Attributes and Directives*, on page 3-66 for more information on adding directives to source code.

Result Analysis

This chapter describes typical analysis tasks. It describes graphical analysis with the HDL Analyst tool as well as interpretation of the text log file. It covers the following:

- Checking Log Results, on page 4-2
- Handling Messages, on page 4-8
- Basic Operations in the Schematic Views, on page 4-16
- Exploring Design Hierarchy, on page 4-30
- Finding Objects, on page 4-37
- Crossprobing, on page 4-48
- Analyzing With the HDL Analyst Tool, on page 4-56
- Analyzing Timing, on page 4-73
- The Island Timing Report, on page 4-83
- The Island Timing Analyst, on page 4-88
- Island Timing Report Critical Paths, on page 4-98

For information about using the Synplify Premier Physical Analyst, see Chapter 5, *Physical Analyst*.

Checking Log Results

You can check the log file for information about the synthesis run. In addition, the Synplify Pro and Synplify Premier interfaces have a Tcl Script window, that echoes each command as it is run. The following describe different ways to check the results of your run:

- Viewing the Log File, next
- Analyzing Results Using the Log File Reports, on page 4-5
- Using the Log Watch Window, on page 4-6

Viewing the Log File

The log file contains the most comprehensive results and information about a synthesis run. The default log file is in HTML format, but there is a text version available too.

For Synplify Pro or Synplify Premier users who just want to check a few critical performance criteria, it is easier to use the Log Watch window (see *Using the Log Watch Window*, on page 4-6) instead of the log file. For details, read through the log file.

1. To view the log file, do one of the following:

- To view the log file in the default HTML format, select View->Log File or click the View Log button in the Project window. You see the log file in HTML format. Alternatively you can double-click the *designName_srr.htm* file in the Implementation Results view to open the HTML log file.
- To see a text version of the log file, double-click the *designName_srr* file in the Implementation Results view. A Text Editor window opens with the log file.

Alternatively, you can set the default to show the text file version instead of the HTML version. Select Options->Project View Options, and toggle off the View log file in HTML option.

The log file lists the compiled files, details of the synthesis run, color-coded errors, warnings and notes, and a number of reports. For infor-

The areas of the log file that are most important are the warning messages and the timing report. The log file includes a timing report that lists the most critical paths. The Synplify Pro and Synplify Premier products also let you generate a report for a path between any two designated points, see *Analyzing Paths with the Timing Analyst*, on page 4-76. The following table lists places in the log file you can use when searching for information.

To find...		Search for...	
Notes		@N or look for blue text	
Warnings and errors		@W and @E, or look for purple and red text respectively	
Performance summary		Performance Summary	
The beginning of the timing report		START TIMING REPORT	
Detailed information about slack times, constraints, arrival times, etc.		Interface Information	
Resource usage		Resource Usage Report	
Gated clock conversions		Gated clock report	

4. Resolve any errors and check all warnings.

You must fix errors, because you cannot synthesize a design with errors. Check the warnings and make sure you understand them. See *Checking Results in the Message Viewer*, on page 4-8 for information. Notes are informational and usually can be ignored. For details about crossprobing and fixing errors, see *Handling Warnings*, on page 4-14, *Editing HDL Source Files with the Built-in Text Editor*, on page 2-5, and *Crossprobing from the Text Editor Window*, on page 4-51.

5. If you are trying to find and resolve warnings, you can bookmark them as shown in this procedure:

- Select Edit->Find or press Ctrl-f.
- Type @W as the criteria on the Find form and click Mark All. The software inserts bookmarks at every line with a warning. You can now page through the file from bookmark to bookmark using the commands in the Edit menu or the icons in the Edit toolbar. For more

- Check the number and types of components used to determine if you have used too much of your resources.

Using the Log Watch Window

The Synplify Pro and Synplify Premier Log Watch window provides a more convenient viewing mechanism than the log file for quickly checking key performance criteria or comparing results from different runs. Its limitation is that it only displays certain criteria. If you need details, use the log file, as described in *Viewing the Log File*, on page 4-2.

1. Open the Log Watch window, if needed, by checking View->Log Watch Window.
- If you open an existing project, the Log Watch window shows the parameters set the last time you opened the window.

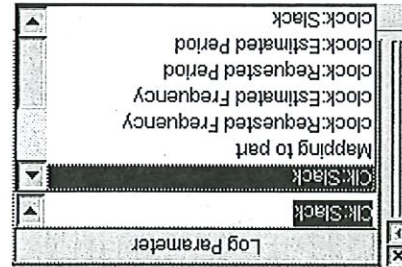
2. If you need a larger window, either resize the window or move the Log Watch window as described below.

- Hold down Ctrl or Shift, click on the window, and move it to a position you want. This makes the Log Watch window an independent window, separate from the Project view.

- To move the window to another position within the Project view, right-click in the window border and select Float in Main Window. Then move the window to the position you want, as described above.

See *Log Watch Window*, on page 2-6 in the *Reference Manual* for information about the popup menu commands.

3. Select the log parameter you want to monitor by clicking on a line and selecting a parameter from the resulting popup menu.



The software automatically fills in the appropriate value from the last synthesis run. You can check the clock requested and estimated

Handling Warnings

If you get warnings (@W prefix) after a synthesis run, do the following:

- Read the warning message and decide if it is something you need to act on, or whether you can ignore it.
- If the message is not self-explanatory or if you are unsure about how to handle the error, click the message ID in either the message window or HTML log file or double click the message ID in the ASCII text log file. These actions take you to online information about the condition that generated the warning.

Automating Message Filtering with a synhooks Script

The following example shows you how to use a synhooks Tcl script to automatically load a message filter file when a project opens and to send email with the messages after a run.

1. Create a message filter file like the following. (See *Filtering Messages in the Message Viewer*, on page 4-10 or *Filtering Messages from the Command Line*, on page 4-13 for details about creating this file.)


```
log_filter -clear
log_filter -hide_matches
log_filter -field report=="VIRTEX2P MAPPER"
log_filter -field type==NOTE
log_filter -field message=="Input *"
log_filter -field message=="Pruning *"
puts "DONE!"
```

2. Copy the synhooks.tcl file and set the environment variable as described in *Automating Flows with synhooks.tcl*, on page 10-10.
3. Edit the synhooks.tcl file so that it reads like the following example. For syntax details, see *Tcl synhooks File Syntax*, on page 5-58 in the *Reference Manual*.
 - The following loads the message filter file when the project is opened. Specify the name of the message filter file you created in step 1. Note that you must source the file.

Basic Operations in the Schematic Views

The RTL and Technology views are schematic views used to graphically analyze your design. In the Synplify product, these views are part of the optional HDL Analyst package. The RTL view is available after a design is compiled; the Technology view is available after a design has been synthesized and contains technology-specific primitives. In the Synplify Premier product, a RTL Floorplan view is available after a floorplan has been created with physical constraint regions and synthesized for the device.

For detailed descriptions of these views, see Chapter 2 of the *Reference Manual*. This section describes basic procedures you use in the RTL and Technology views. The information is organized into these topics:

- Differentiating Between the Views, next
- Opening the Views, on page 4-17
- Analyzing Your Design Graphically, on page 4-19
- Viewing Object Properties, on page 4-20
- Selecting Objects in the RTL/Technology Views, on page 4-23
- Working with Multisheet Schematics, on page 4-24
- Moving Between Views in a Schematic Window, on page 4-26
- Setting Schematic View Preferences, on page 4-26
- Managing Windows, on page 4-28

For information on specific tasks like analyzing critical paths, see the following sections:

- Exploring Object Hierarchy by Pushing/Popping, on page 4-31
- Exploring Object Hierarchy of Transparent Instances, on page 4-36
- Browsing to Find Objects, on page 4-37
- Crossprobing, on page 4-48
- Analyzing With the HDL Analyst Tool, on page 4-56
- Analyzing Timing, on page 4-73

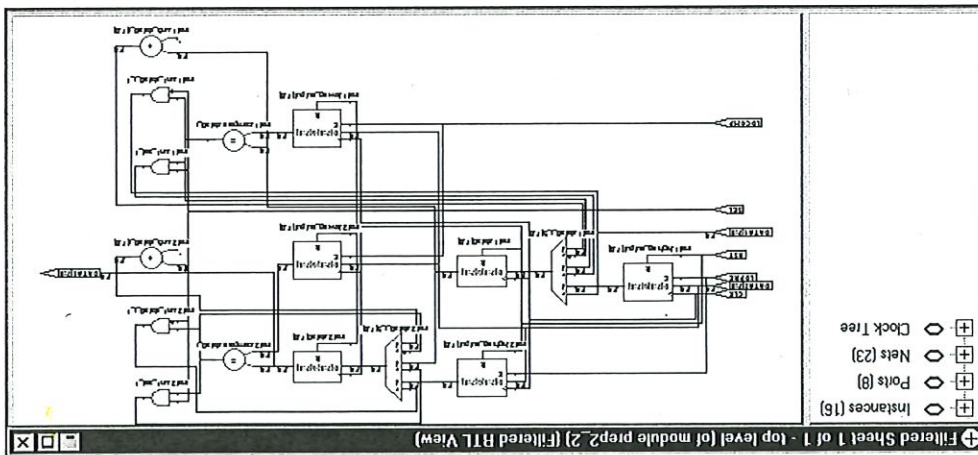
Differentiating Between the Views

- The difference between the RTL and Technology views is that the RTL view is the view generated after compilation, while the Technology view is the view generated after mapping. The RTL view displays your design as a high-level, technology-independent schematic. At this high level of abstraction, the design is represented with technology-independent components like variable-width adders, registers, large muxes, state machines, and so on. This view corresponds to the .srs netlist file generated by the software in the Synplicity proprietary format. For a detailed description, see Chapter 2 of the *Reference Manual*.
- The Technology view contains technology-specific primitives. It shows low-level, vendor-specific components such as look-up tables, cascade and carry chains, muxes, and flip-flops, which can vary with the vendor and the technology. This view corresponds to the .srn netlist file, generated by the software in the Synplicity proprietary format. For a detailed description, see Chapter 2 of the *Reference Manual*.
- The Synplify Premier RTL Floorplan view displays a floorplan schematic that includes all the logic assigned to any physical constraint regions created on the device, as well as, all other logic of the design. This view uses the same high-level abstraction and technology-independent components of the RTL view.

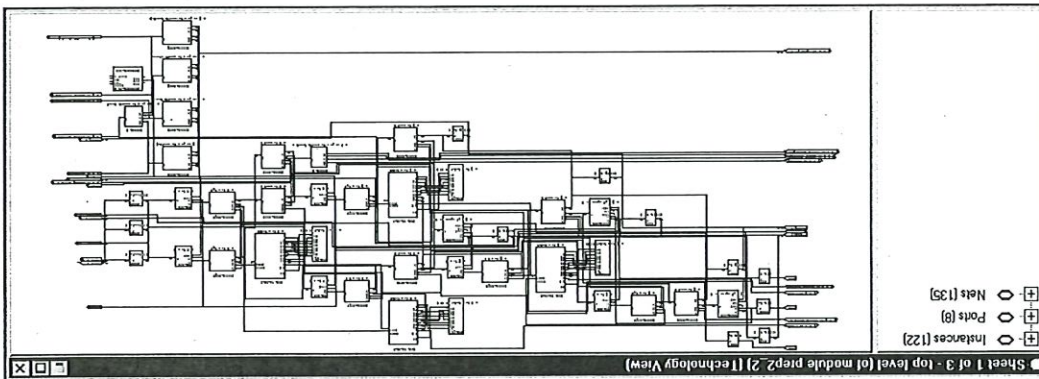
Opening the Views

The procedure for opening an RTL or Technology view is similar; the main difference is the design stage at which these views are available.

RTL View



Technology View



Analyzing Your Design Graphically

By using BEST® (Behavior Extraction Synthesis Technology) in the RTL view, the software keeps a high-level of abstraction and makes the RTL view easy to view and debug. High-level structures like RAMs, ROMs, operators, and FSMs are kept as abstractions in this view instead of being converted to gates. You can examine the high-level structure, or push into a component and view the gate-level structure.

In the Technology view, the software uses module generators to implement the high-level structures from the RTL view using technology-specific resources.

Exploring Design Hierarchy

Schematics generally have a certain amount of design hierarchy. You can move between hierarchical levels using the Hierarchy Browser or Push/Pop mode. For additional information, see *Analyzing With the HDL Analyst Tool*, on page 4-56. The topics include:

- Traversing Design Hierarchy with the Hierarchy Browser, on page 4-30
- Exploring Object Hierarchy by Pushing/Popping, on page 4-31
- Exploring Object Hierarchy of Transparent Instances, on page 4-36

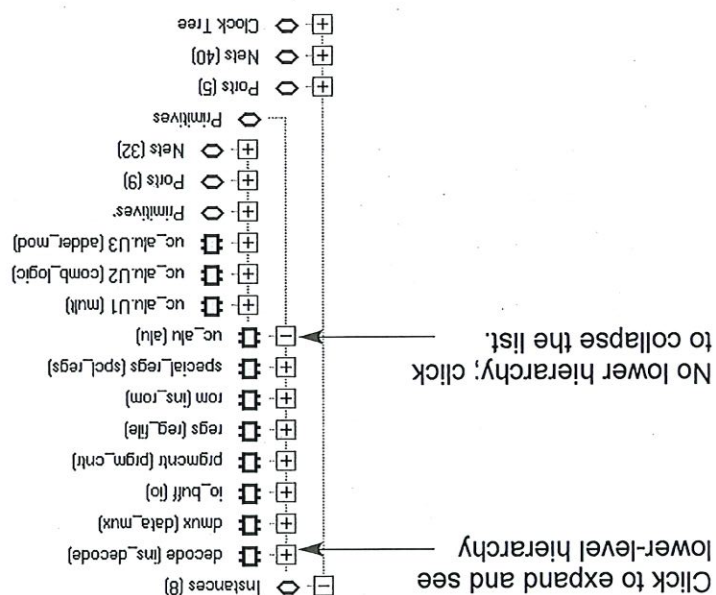
Traversing Design Hierarchy with the Hierarchy Browser

The Hierarchy Browser is the list of objects on the left side of the RTL and Technology views. It is best used to get an overview, or when you need to browse and find an object. If you want to move between design levels of a particular object, Push/Pop mode is more direct. Refer to *Exploring Object Hierarchy by Pushing/Popping*, on page 4-31 for details.

The hierarchy browser allows you to traverse and select the following:

- Instances and submodules
- Ports
- Internal nets
- Clock trees (in an RTL view)

The browser lists the objects by type. A plus sign in a square icon indicates that there is hierarchy under that object and a minus sign indicates that the design hierarchy has been expanded. To see lower-level hierarchy, click on the plus sign for the object. To ascend the hierarchy, click on the minus sign.



Refer to *Hierarchy Browser Symbols*, on page 2-19 in the *Reference Manual* for an explanation of the symbols.

Exploring Object Hierarchy by Pushing/Popping

To view the internal hierarchy of a specific object, it is best to use Push/Pop mode or examine transparent instances, instead of using the Hierarchy Browser described in *Traversing Design Hierarchy with the Hierarchy Browser*, on page 4-30. You can access Push/Pop mode with the Push/Pop Hierarchy icon, the Push/Pop Hierarchy command, or mouse strokes.

When combined with other commands like filtering and expansion commands, Push/Pop mode can be a very powerful tool for isolating and analyzing logic. See *Filtering Schematics*, on page 4-60, *Expanding Pin and Net Logic*, on page 4-62, and *Expanding and Viewing Connections*, on page 4-66 for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

- Pushing into Objects, next
- Popping up a Hierarchical Level, on page 4-35

Analyzing Timing

You can use the Timing Analyst and HDL Analyst functionality to analyze timing. This section describes the following:

- Analyzing Clock Trees in the RTL View, next
- Viewing Critical Paths, on page 4-74
- Analyzing Paths with the Timing Analyst, on page 4-76
- Analyzing Paths with the Synplify Premier Timing Analyst, on page 4-79
- Handling Negative Slack, on page 4-81

Synplify Premier users can also use island timing reports for analysis, as described in *The Island Timing Report*, on page 4-83.

Analyzing Clock Trees in the RTL View

To analyze clock trees in the RTL view:

1. In the Hierarchy Browser, expand Clock Tree, select all the clocks, and filter the design.
 - The Hierarchy Browser lists all clocks and the instances that drive them under Clock Tree. The filtered view shows the selected objects.
 2. If necessary, use the filter and expand commands to trace clock connections back to the ports and check them.
 - For details about the commands for filtering and expanding paths, see *Filtering Schematics*, on page 4-60, *Expanding Pin and Net Logic*, on page 4-62 and *Expanding and Viewing Connections*, on page 4-66. For more information on filtering schematics, see *HDL Analyst Menu: Filtering and Flattening Commands*, on page 3-73 of the *Reference Manual*.
 3. Check that your defined clock constraints cover the objects in the design.
- If you do not define your clock constraints accurately, you might not get the best possible synthesis optimizations.

Viewing Critical Paths

The HDL Analyst tool makes it simple to find and examine critical paths and the relevant source code. The following procedure shows you how to filter and analyze a critical path. You can also use the procedure described in *Analyzing Paths with the Timing Analyst*, on page 4-76 and *Analyzing Paths with the Synplify Premier Timing Analyst*, on page 4-79 to view this and other paths.

1. If needed, set the slack time for your design.
 - Select HDL Analyst->Set Slack Margin.
 - To view only instances with the worst-case slack time, enter a zero.
 - To set a slack margin range, type a value for the slack margin, and click OK. The software gets a range by subtracting this number from the slack time, and the Technology view displays instances within this range. For example, if your slack time is -10 ns, and you set a slack margin of 4 ns, the command displays all instances with slack times between -6 ns and -10 ns. If your slack margin is 6 ns, you see all instances with slack times between -4 ns and -10 ns.

2. Display the critical path using one of the following methods. The Technology view displays a hierarchical view that highlights the instances and nets in the most critical path of your design.
 - To generate a hierarchical view of the critical path, click the Show Critical Path icon (stopwatch icon ) , select HDL Analyst->Technology->Hierarchical Critical Path, or select the command from the popup menu. This is a filtered view in the same window, with hierarchical logic shown in transparent instances. History commands apply, so you can return to the previous view by clicking Back.
 - To flatten the hierarchical critical path described above, right-click and select Flatten Schematic. The software generates a new view in the current window, and flattens only the transparent instances needed to show the critical path; the rest of the design remains hierarchical. Click Back to go the top-level design.
 - To generate a flattened critical path in a new window, select HDL Analyst->Technology->Flattened Critical Path. This command uses more memory because it flattens the entire design and generates a new view for the flattened critical path in a new window. Click Back in this window to go to the flattened top-level design or to return to the previous window.

6. Click the Back icon to return to the previous view. If you flattened your design during analysis, select Unflatten Schematic to return to the top-level design.
- There is no need to regenerate the critical path, unless you flattened your design during analysis or changed the slack margin. When you flatten your design, the view is regenerated so the history commands do not apply and you must click the Critical Path icon again to see the critical path view.
7. Return synthesis, and check your results.
- If you have fixed the path, the window displays the next most critical path when you click the icon.
- Repeat this procedure and fix the design for the remaining critical paths. When you are within 5-10 percent of your desired results, place and route your design to see if you meet your goal. If so, you are done. If your vendor provides timing-driven place and route, you might improve your results further by adding timing constraints to place and route.

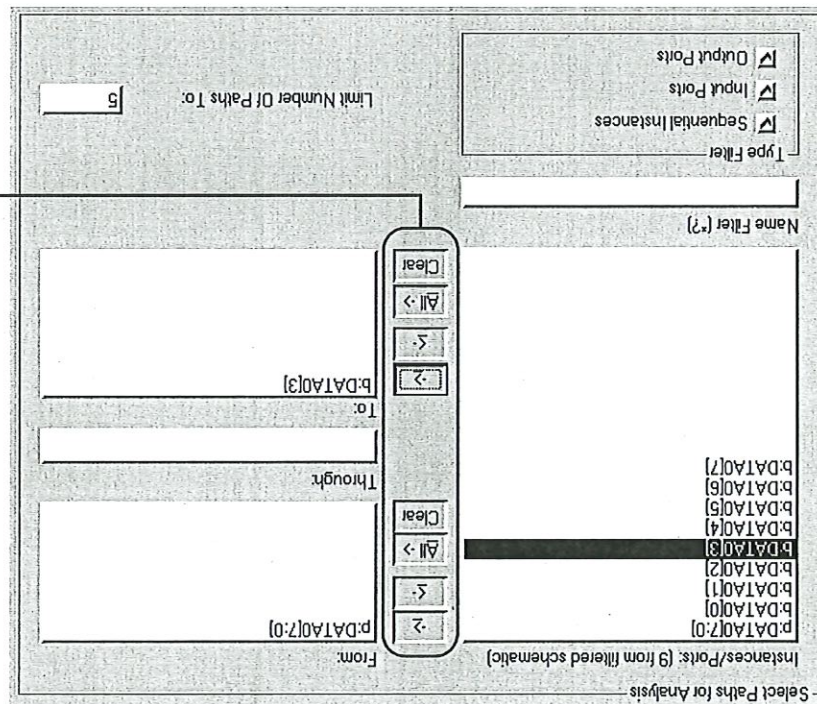
Analyzing Paths with the Timing Analyst

The Synplify Pro and Synplify Premier Timing Analyst is an analysis tool that lets you analyze the timing between any two points in applicable Actel, Altera and Xilinx technologies, without resynthesizing your design. You can use this tool for any path, but for critical paths it is easier to use the procedure described in *Viewing Critical Paths*, on page 4-74. The following procedure describes how to use the Timing Analyst; see *Analyzing Paths with the Synplify Premier Timing Analyst*, on page 4-79 for information about the Synplify Premier tool.

1. Open a Technology view. You can only analyze paths in a mapped design.
 2. Select instances from the schematic or use Find to find and select start and end registers or ports. You can filter or expand your design to find start and end points at different levels of hierarchy.
 3. Open the Timing Analyst window by clicking the icon  or by selecting HDL Analyst->Timing Analyst.
- The Timing Analyst window opens with the selected start and end points in the box on the left side. If you did not select start and end points, it shows all the signal names. For detailed descriptions of the options in

this window, see *Timing Analyst Command*, on page 3-80 in the *Reference Manual*.

4. Select the objects and move them into the start and end point boxes using the appropriate arrows.

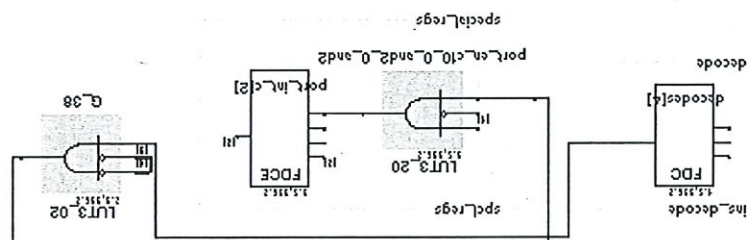


5. Click Generate.

The software generates and opens a timing report and a timing view schematic. The timing report (.ta file in the Implementation Results view) contains from-to information for just the path you specified, and is different from the timing report for the entire design that is in the log file. The timing view schematic (.ta.srm) is a filtered Technology view that shows the path between the start and end points. You cannot generate a critical path or use the Timing Analyst from this view. If you close the report and schematic windows, reopen them by selecting the .ta (timing report) and name.ta.srm (Timing view) files from the Implementation Results view.

6. View the results.

The following figure shows a timing view for a path; the file excerpt that follows shows associated details from the timing report.



Worst From-To Paths Information

Path information for path number 1:

Requested Period: 1000.000
- Setup time: 0.370
- Required time: 999.630
- Propagation time: 3.456
= Slack: 996.174

Starting point: decode_decode[4] / 0
Ending point: special_regs.port_int_c[2] / CR
The start point is clocked by eight_bit_clock [rising] on pin C
The end point is clocked by eight_bit_clock [rising] on pin C

Instance/Net		Pin	Pin	Type	Name	Dir	Delay	Arrival	Fan
Name									
decode_decode[4]	RDC	Q	Out		1.472				14
Net									
LUT3	IO	In			1.472				
LUT3	0	Out			0.863				
Net									
special_regs.port_en_c10_and2_0_and2									
LUT3	IO	In			2.336				3
Net									
special_regs.port_en_c10_and2_0_and2									
LUT3	0	Out			1.120				
Net									
special_regs.port_int_c[2]									
Net									
special_regs.port_int_c[2]									
RDC	CR	In			3.456				8

7. Use the HDL analyst commands described in *Analyzing With the HDL Analyst Tool*, on page 4-56 to analyze the path.

Synplify Premier Physical Analyst Tool

The Physical Analyst tool provides a visual display of the placement and global routing of the design after running place and route with backannotation.

The Physical Analyst is available in each of the Synplify Premier flows with the following criteria:

- Graph-based Physical Synthesis—Physical Synthesis is enabled without a Design Plan or place-and-route implementation. The tool automatically performs placement with backannotation during the physical synthesis run. For Xilinx Virtex-II Pro, Virtex-4, and Spartan-3.
- Graph-based Physical Synthesis with a Design Plan—Physical Synthesis is enabled with a Design Plan, but without a place-and-route implementation. The tool automatically performs placement with backannotation during the physical synthesis run. For Xilinx Virtex-II Pro, Virtex-4, and Spartan-3.
- Design-plan based Physical Synthesis—Physical Synthesis is enabled with a Design Plan and a place-and-route implementation created with backannotation enabled. For Altera Cyclone, Cyclone-II, Stratix, Stratix-GX, Stratix-II, and Xilinx Virtex-II.

The Physical Analyst includes the following capabilities:

- Display placement information such as cell locations and signal pins.
- Analyze netlists using various commands, such as filter, show critical path, or route all nets.
- Query object and properties of instances and nets.
- Cross probe between the Synplify Premier Physical Analyst and either the HDL Analyst or the source code text file.

This section describes basic procedures you use in the Physical Analyst view. The information is organized into these topics:

- Opening the Physical Analyst View, on page 5-4
- Using the Physical Analyst Control Panel, on page 5-5

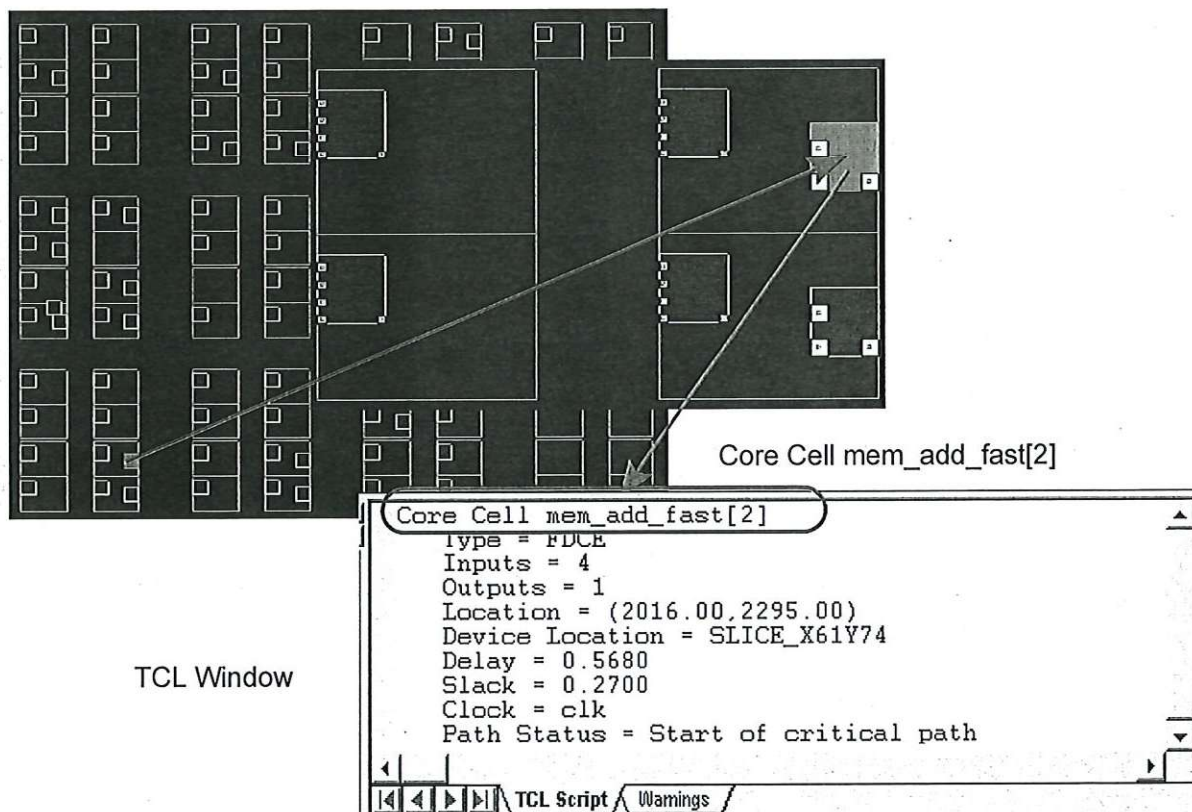
The Physical Analyst view uses the following input files:

Transcribing Object Selections

The Selection Transcription command allows you to select an object in the Physical Analyst view, then display the object's tool tip information in the TCL window. To enable or disable this command, either:

- Right-click in the Physical Analyst view and select Selection Transcription from the popup menu.
- Select the View->Selection Transcription menu option.

The Selection Transcription command is enabled by default. Transcription only occurs when a single object is selected. Do not use transcription with area selections or when objects are selected using other commands such as Expand or Find. The following figure shows an example of an object selected on the device and its tool tip information displayed in the TCL window.



You can also use the information from the TCL window display to copy and paste into other windows or files such as SCOPE, the Find Object dialog box, or a text file.

Analyzing Timing with the Physical Analyst

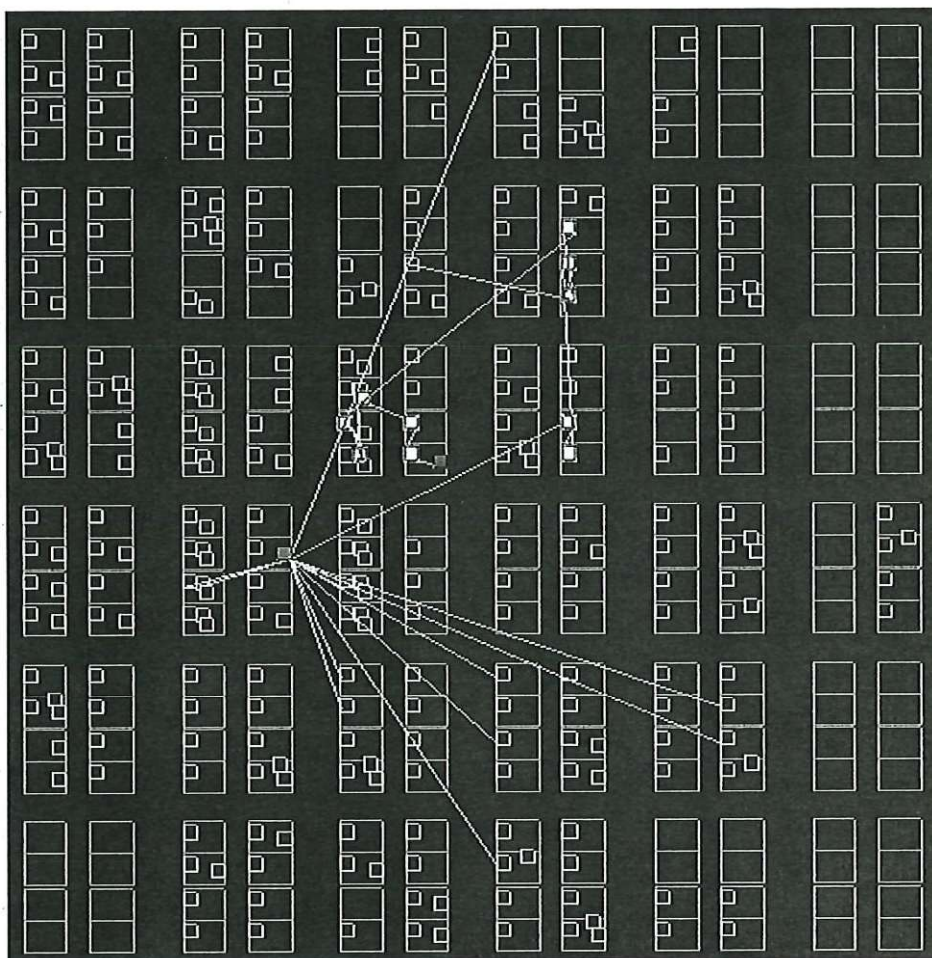
You can use the Physical Analyst functionality to analyze timing. This section describes how to view and use the critical path for further physical synthesis.

- Viewing Critical Paths, on page 5-51
- Tracing Critical Paths Forward and Backwards, on page 5-54

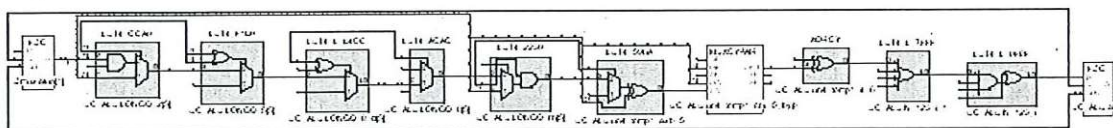
Viewing Critical Paths

The Physical Analyst tool makes it easy to find and examine critical paths and the relevant logic in the HDL Analyst schematic view. Make sure the HDL Analyst view is open, for example, by selecting HDL Analyst->Technology->Flattened View or HDL Analyst->RTL->Flattened View. The following procedure shows you how to filter and analyze a critical path.

1. To generate a view of the critical path with the Physical Analyst tool, click the Show Critical Path icon (stopwatch icon ) or select the command from the popup menu. To zoom in on the critical path, right-click and select Zoom Selected from the popup menu.



2. Check the Technology view. Click the Filter on Selected Gates icon () to display the critical path.



3. You can also cross probe the critical path from the flattened Technology view to the Physical Analyst view by clicking on the Show Critical Path icon (). Then, right-click and select Select All Schematic->Instances. Make sure the Physical Analyst view is open.
4. Check the Physical Analyst view. Critical path instances and nets should be highlighted in this view. See the figure in step 1.

CHAPTER 6

Design Optimization

This chapter covers techniques for optimizing your design using built-in tools or attributes. For vendor-specific optimizations, see Chapter 8, *Vendor-Specific Optimizations*.

It describes the following:

- Design Guidelines, on page 6-2
- Optimizing Results, on page 6-5
- Defining State Machines for Synthesis, on page 6-13
- Using the Symbolic FSM Compiler, on page 6-17
- Using FSM Explorer, on page 6-22
- Using the FSM Viewer, on page 6-25
- Defining Black Boxes for Synthesis, on page 6-30
- Pipelining, on page 6-40
- Retiming, on page 6-44
- Inserting Probes, on page 6-50
- Inferring RAMs, on page 6-54
- Inferring Shift Registers, on page 6-80
- Forward Annotation of Initial Values
- Working with LPMs, on page 6-87
- Working with Gated Clocks, on page 6-99

Design Guidelines

The software automatically makes efficient tradeoffs to achieve the best results. However, you can optimize your results by using the appropriate control parameters. This section describes general design guidelines for optimization. The topics have been categorized as follows:

- General Optimization Tips, next
- Area Optimization Tips, on page 6-3
- Timing Optimization Settings, on page 6-4

General Optimization Tips

This section contains general optimization tips that are not directly area or timing-related. For area optimization tips, see *Area Optimization Tips*, on page 6-3. For timing optimization, see *Timing Optimization Settings*, on page 6-4.

- In your source code, remove any unnecessary priority structures in timing-critical designs. For example, use CASE statements instead of nested IF-THEN-ELSE statements for priority-independent logic.
- If your design includes safe state machines, use the `syn_encoding` attribute with a value of `safe`. This ensures that the synthesized state machines never lock in an illegal state.
- For FSMs coded in VHDL using enumerated types, use the same encoding style (`syn_enum_encoding` attribute value) on both the state machine enumerated type and the state signal. This ensures that there are no discrepancies in the type of encoding to negatively affect the final circuit.
- Make sure that the source code supports inferencing or instantiation by using architecture-specific resources like memory blocks.
- Some designs benefit from hierarchical optimization techniques. To enable hierarchical optimization on your design, set the `syn_hier` attribute to `firm`.
- For accurate results with timing-driven synthesis, explicitly define clock frequencies with a constraint, instead of using a global clock frequency.

Area Optimization Tips

This section contains information on optimizing to reduce area. Optimizing for area often means larger delays, and you will have to weigh your performance needs against your area needs to determine what works best for your design. For tips on optimizing for performance, see *Timing Optimization Settings*, on page 6-4. General optimization tips are in *General Optimization Tips*, on page 6-2.

- Increase the fanout limit when you set the implementation options. A higher limit means less replicated logic and fewer buffers inserted during synthesis, and a consequently smaller area. In addition, as P&R tools typically buffer high fanout nets, there is no need for excessive buffering during synthesis. See *Setting Fanout Limits*, on page 6-7 for more information.
- Check the Resource Sharing option when you set implementation options. With this option checked, the software shares hardware resources like adders, multipliers, and counters wherever possible, and minimizes area. See *Sharing Resources*, on page 6-5 for details.
- For designs with large FSMs, use the gray or sequential encoding styles, because they typically use the least area. For details, see *Specifying FSMs with Attributes and Directives*, on page 6-15.
- If you are mapping into a CPLD and do not meet area requirements, set the default encoding style for FSMs to sequential instead of onehot. For details, see *Specifying FSMs with Attributes and Directives*, on page 6-15.
- For small CPLD designs (less than 20K gates), you might improve area by using the `syn_hier` attribute with a value of `flatten`. When specified, the software optimizes across hierarchical boundaries and creates smaller designs.

Timing Optimization Settings

This section contains information on optimizing to meet timing requirements. Optimizing for timing is often at the expense of area, and you will have to balance the two to determine what works best for your design. For tips on optimizing for area, see *Area Optimization Tips*, on page 6-3. General optimization tips are in *General Optimization Tips*, on page 6-2.

- Use realistic design constraints, about 10 - 15% of the real goal. Over constraining your design can be counter-productive because you can get poor implementations. Use clock, false path, and multicycle path constraints to make the constraints realistic.
- Select a balanced fanout constraint. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated logic. See *Setting Fanout Limits*, on page 6-7 for information about setting limits.
- If the critical path goes through arithmetic components, try disabling Resource Sharing. You can get faster times at the expense of increased area, but use this technique carefully. Adding too many resources can cause longer delays and defeat your purpose.
- If the P&R and synthesis tools report different critical paths, use a timing constraint with the `-route` option. With this option, the software adds route delay to its calculations when trying to meet the clock frequency goal. Use realistic values for the constraints.
- For FSMs, use the onehot encoding style, because it is often the fastest implementation. If a large output decoder follows an FSM, gray or sequential encoding could be faster.
- For designs with black boxes, characterize the timing models accurately, using the `syn_tpd`, `syn_tco`, and `syn_tso` directives.
- If you saw warnings about feedback muxes being created for signals when you compiled your source code, make sure to assign set/resets for the signals. This improves performance by eliminating the extra mux delay on the input of the register.
- Make sure that you pass your timing constraints to the place-and-route tools, so that they can use the constraints to optimize timing.

Optimizing Results

You can optimize your results with attributes and directives, some of which are specific to the technology you are using. Similarly, you can use specify objects or hierarchy that you want to preserve during synthesis. For a complete list of all the directives and attributes, see the *Reference Manual*. This section describes the following:

- Sharing Resources, next
- Setting Fanout Limits, on page 6-7
- Controlling Buffering and Replication, on page 6-8
- Controlling Hierarchy Flattening, on page 6-10
- Preserving Objects from Optimization, on page 6-10
- Preserving Hierarchy, on page 6-12

Sharing Resources

One of the ways you can optimize area is to use resource sharing. With resource sharing, the software uses the same arithmetic operators for mutually exclusive statements; for example, with the branches of a case statement. Conversely, you can improve timing by disabling resource sharing, but at the expense of increased area.

1. Specify resource sharing globally for the whole design with one of the methods below. Enable the option to improve area; disable it to improve timing.
 - Select Project->Implementation Options->Options, and enable or disable Resource Sharing. Alternatively, enable Resource Sharing in the Project view.
 - Apply the `syn_sharing` directive to the top-level module or architecture in the source code. See *syn_sharing Directive*, on page 8-181 of the *Reference Manual* for syntax examples.

```
Verilog module top(out, in, clk_in) /* synthesis syn_sharing = "on" */;
```

```
VHDL architecture rtl of top is
    attribute syn_sharing : string;
    attribute syn_sharing of rtl : architecture is "off";
```

Setting Fanout Limits

Optimization affects net fanout. If your design has critical nets with high fanout, you can set fanout limits. You can only do this in certain technologies. For details specific to individual technologies, see the *Reference Manual*.

1. To set a global fanout limit for the whole design, do either of the following:
 - Select Project-> Implementation Options->Device and type a value for the Fanout Guide option.
 - Apply the `syn_maxfan` attribute to the top-level view or module.

The value sets the number of fanouts for a given driver, and affects all the nets in the design. The defaults vary, depending on the technology. Select a balanced fanout value. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated or buffered logic. Both extremes affect routing and design performance. The right value depends on your design. The same value of 32 might result in fanouts of 11 or 12 and large delays on the critical path in one design or in excessive replication in another design.

The software uses the value as a soft limit, or a guide. It traverses the inverters and buffers to identify the fanout, and tries to ensure that all fanouts are under the limit by replicating or buffering where needed (see *Controlling Buffering and Replication*, on page 6-8 for details). However, the synthesis tool does not respect the fanout limit absolutely; it ignores the limit if the limit imposes constraints that interfere with optimization.

2. For certain Actel technologies, you can set a global hard fanout limit by doing the following:
 - Select Project-> Implementation Options->Device and type a value for the Fanout Guide option, as described in the previous step.
 - On the same tab, check the Hard Fanout Limit option.

This makes the specified value a global hard fanout limit for the design.

3. To override the global fanout guideline and set a soft fanout limit at a lower level, set the `syn_maxfan` attribute on modules, views, or non-primitive instances.

These limits override the more global limits for that object (including a global hard limit in Actel technologies). However, these limits still

Controlling Hierarchy Flattening

Optimization flattens hierarchy. To control the flattening, use the `syn_hier` attribute as described here. You can also use the attribute to prevent flattening, as described in *Preserving Hierarchy*, on page 6-12.

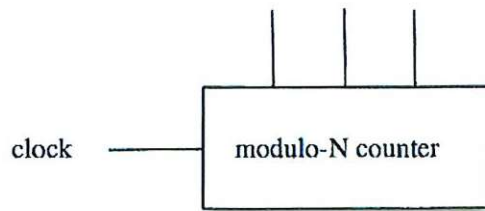
1. Attach the `syn_hier` attribute to the module or architecture you want to preserve. You can also add the attribute in SCOPE. If you use SCOPE to enter the attribute, make sure to use the `v:` syntax.
2. Set the attribute value:

To...	Value...
Flatten all levels below, but not the current level	flatten
Remove the current level of hierarchy without affecting the lower levels	remove
Remove the current level of hierarchy and the lower levels	flatten, remove
Flatten the current level (if needed for optimization)	soft

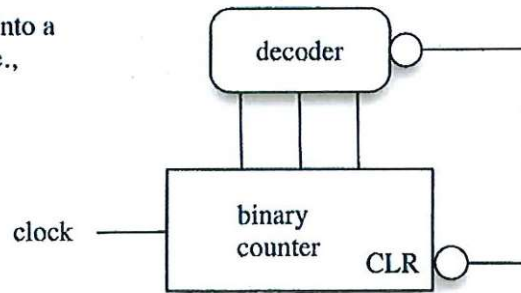
The software flattens the design as directed. If there is a lower-level `syn_hier` attribute, it takes precedence over a higher-level one.

Preserving Objects from Optimization

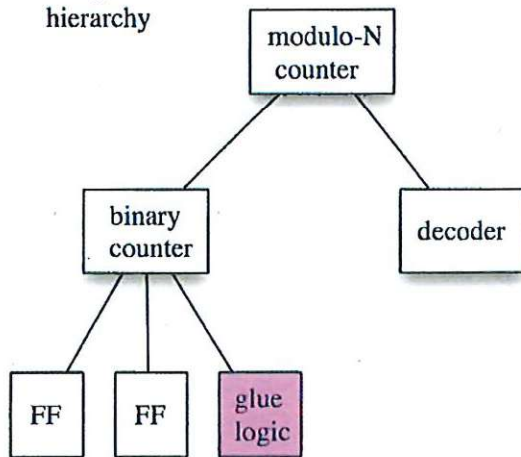
Synthesis can collapse or remove nets during optimization. If you want to retain a net for simulation, probing, or for a different synthesis implementation, you must specify this with an attribute. Similarly, the software removes duplicate registers or instances with unused output. If you want to preserve this logic for simulation or analysis, you must use an attribute. The following table lists the attributes to use in each situation. For details about the attributes and their syntax, see the *Reference Manual*.



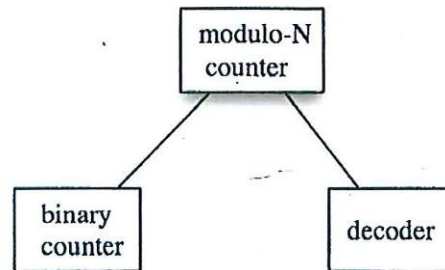
partitioning divides the counter into a binary counter and a decoder (i.e., smaller blocks of functionality)



design hierarchy



design flattening removes the hierarchy by merging modules (i.e., removing the hierarchy)



Design flattening makes it easier to optimize the circuitry. But the larger modules are less portable and reusable.

Specifying FSMs with Attributes and Directives

If your design has state machines, the software can extract them automatically with the FSM Compiler (see *Using the Symbolic FSM Compiler*, on page 6-17), or you can manually specify attributes to define the state machines. You attach the attributes to the state registers. For detailed information about the attributes and their syntax, see the *Reference Manual*.

The following steps show you how to use attributes to define FSMs for extraction. For alternative ways to define state machines, see *Defining State Machines in Verilog*, on page 6-13 and *Defining State Machines in VHDL*, on page 6-14.

1. To determine how state machines are extracted, set attributes in the source code as shown in the following table:

To...	Attribute
Specify a state machine for extraction and optimization	<code>syn_state_machine=1</code>
Prevent state machines from being extracted and optimized	<code>syn_state_machine=0</code>
Prevent the state machine from being optimized away	<code>syn_preserve=1</code>

For information about how to add attributes, see *Adding Attributes and Directives*, on page 3-66.

2. To determine the encoding style used for the state machine, set the `syn_encoding` attribute in the source code or in the SCOPE window. For VHDL users there are alternative methods, described in the next step.

The FSM Compiler and the FSM Explorer honor this setting. The different values for this attribute are briefly described here:

Situation: If...	syn_encoding Value	Explanation
Area is important	sequential	One of the smallest encoding styles.
Speed is important	onehot	Usually the fastest style and suited to most FPGA styles.
Recovery from an invalid state is important	safe, with another style. For example: /* synthesis syn_encoding = "safe, onehot" */	Forces the state machine to reset. For example, where an alpha particle hit in a hostile operating environment causes a spontaneous register change, you can use safe to reset the state machine.
There are <5 states	sequential	Default encoding.
Large output decoder follows the FSM	sequential or gray	Could be faster than onehot, even though the value must be decoded to determine the state. For sequential , more than one bit can change at a time; for gray , only one bit changes at a time, but more than one bit can be hot.
There are a large number of flip-flops	onehot	Fastest style, because each state variable has one bit set, and only one bit of the state register changes at a time.

3. If you are using VHDL, you have two choices for defining encoding:

- Use `syn_encoding` as described above, and enable the FSM compiler.
- Use `syn_enum_encoding` to define the states (**sequential**, **onehot**, **gray**, and **safe**) and disable the FSM compiler. If you do not disable the FSM compiler, the `syn_enum_encoding` values are not implemented. This is because the FSM compiler, a mapper operation, overrides `syn_enum_encoding`, which is a compiler directive.

Use this method for user-defined FSM encoding. For example:

```
attribute syn_enum_encoding of state_type : type is "001 010 101";
```


Using the Symbolic FSM Compiler

The Symbolic FSM Compiler is an advanced state machine optimizer, which automatically recognizes state machines in your design and optimizes them. Unlike other synthesis tools that treat state machines as regular logic, the FSM Compiler extracts the state machines as symbolic graphs, and then optimizes them by re-encoding the state representations and generating a better logic optimization starting point for the state machines. The FSM Explorer uses the state machines extracted by the FSM Compiler when it explores different encoding styles. The FSM Explorer option is only available in the Synplify Pro and Synplify Premier tools.

For more information, see the following:

- Choosing When to Use the FSM Compiler, on page 6-17, next
- Running the FSM Compiler on the Whole Design, on page 6-18
- Running the FSM Compiler on Individual FSMs, on page 6-20
- Specifying FSMs with Attributes and Directives, on page 6-15

Choosing When to Use the FSM Compiler

The FSM Compiler and the FSM Explorer are automatic tools for state machines, but you can also specify FSMs manually with attributes. For more information about the FSM Explorer and FSM attributes, see *Using FSM Explorer*, on page 6-22, *Adding Attributes and Directives*, on page 3-66 and *Specifying FSMs with Attributes and Directives*, on page 6-15.

Here are the main reasons to use the FSM Compiler:

- To generate better results for your state machines

The software uses optimization techniques that are specifically tuned for FSMs, like reachability analysis for example. The FSM Compiler also lets you convert an encoded state machine to another encoding style (to improve speed and area utilization) without changing the source. For example, you can use a onehot style to improve results.

- To debug the state machines

State machine description errors result in unreachable states, so if you have errors, you will have fewer states. You can check whether your

source code describes your state machines correctly. You can also use the FSM Viewer to see a high-level bubble diagram and crossprobe from there. The FSM Viewer is only available in the Synplify Pro and Synplify Premier tools. For information about the FSM Viewer, see *Using the FSM Viewer*, on page 6-25.

- To run the FSM Explorer

The FSM Explorer is a tool that examines all the encoding styles before selecting the best option, based on the state machine extraction done by the FSM Compiler. If the FSM Compiler has not been run previously, the Explorer automatically runs it. For more information about using the FSM Explorer, see *Using FSM Explorer*, on page 6-22.

Running the FSM Compiler on the Whole Design

1. Enable the compiler by checking the Symbolic FSM Compiler box in one of these places:
 - The main panel on the left side of the project window
 - The Options tab of the dialog box that comes up when you click the New Impl or Impl Options buttons
2. To set a specific encoding style for a state machine, define the style with the `syn_encoding` attribute, as described in *Specifying FSMs with Attributes and Directives*, on page 6-15.

If you do not specify a style, the FSM Compiler picks an encoding style based on the number of states.

3. Click Run to run synthesis.

The software automatically recognizes and extracts the state machines in your design, and instantiates a state machine primitive in the netlist for each FSM it extracts. It then optimizes all the state machines in the design, using techniques like reachability analysis, next state logic optimization, state machine re-encoding and proprietary optimization

Pipelining

The pipelining feature is only available in the Synplify Pro and Synplify Premier tools. Pipelining is the process of splitting logic into stages so that the first stage can begin processing new inputs while the last stage is finishing the previous inputs. This ensures better throughput and faster circuit performance. If you are using selected Altera or Xilinx technologies, you can use or the related technique of retiming to improve performance. See *Retiming*, on page 6-44 for details.

For pipelining, The software splits the logic by moving registers into the multiplier or ROM:

This section discusses the following pipelining topics:

- Prerequisites for Pipelining, next
- Pipelining the Design, on page 6-41

Prerequisites for Pipelining

The pipelining feature is only available in the Synplify Pro and Synplify Premier tools.

- Currently, pipelining is only supported for certain Altera and Xilinx technologies.
- In Xilinx Virtex designs, you can pipeline ROMs and multipliers. In Altera designs, you can pipeline multipliers, but not ROMs.
- For Xilinx Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-4 devices, you can only pipeline multipliers if the adjacent register has a SYNCHRONOUS reset.
- ROMs to be pipelined must be at least 512 words. Anything below this limit is too small.
- For Xilinx Virtex designs, you can push any kind of flip-flop into the module, as long as all the flip-flops in the pipeline have the same clock, the same set/reset signal or lack of it, and the same enable control or lack of it. For Altera designs, you must have asynchronous set/resets if you want to do pipelining.

Pipelining Combinational Logic Blocks

1 Non-Pipelined

```
module comb_mult_add_16 (A, B, C, Y);  
input  [15:0] A, B, C;  // data input  
output [31:0] Y;        // 32-bit output Y  
  
assign Y = (A * B) + C;  
  
endmodule
```

2 Pipelined

```
module mult_add_pipeline_2stage (clk, A, B, C, Y);
input  clk;
input  [15:0] A, B, C;    // 16-bit data inputs
output [31:0] Y           // 32-bit output Y

// Stage 1 pipeline registers
reg [31:0] mult_reg;    // product of A*B
reg [15:0] C_reg;       // pipeline register for C

// Stage 2 pipeline register
reg [31:0] add_reg;

always @(posedge clk)
begin
    // Stage 1: multiply and latch inputs
    mult_reg <= A * B;
    C_reg    <= C;

    // Stage 2: add product + pipelined C
    add_reg  <= mult_reg + C_reg; // C_reg zero-extends to 32 bits
end

assign Y = add_reg;
endmodule
```

Stage 1:

mult_reg holds the product of $A * B$.
C_reg holds the pipelined copy of C.

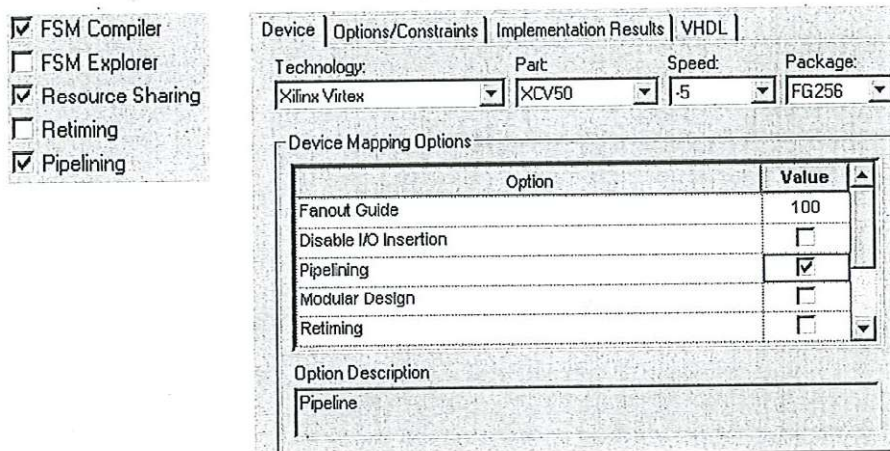
Stage 2:

add_reg stores the sum of mult_reg + C_reg.

Pipelining the Design

The following procedure shows you techniques for pipelining.

1. Make sure the design meets the criteria described in *Prerequisites for Pipelining*, on page 6-40.
2. To enable pipelining for the whole design, check the Pipelining check box, from the button panel in the Project window, or with the Project->Implementation Options command (Device tab). The option is only available in the appropriate technologies.



Use this approach as a first pass to get a feel for which modules you can pipeline. If you know exactly which registers you want to pipeline, add the attribute to the registers in the source code or interactively using the SCOPE interface.

3. To check whether individual registers are suitable for pipelining, do the following:
 - Open the RTL view of the design.
 - Select the register and press F12 to filter the schematic view.

Retiming

The retiming feature is only available in the Synplify Pro and Synplify Premier tools. Retiming is a powerful technique for improving the timing performance of sequential circuits without having to modify the source code. Retiming automatically moves registers (register balancing) across combinatorial gates or LUTs to improve timing while ensuring identical behavior as seen from the primary inputs and outputs of the design. Retiming moves registers across gates or LUTs, but does not change the number of registers in a cycle or path from a primary input to a primary output. However, it can change the total number of registers in a design.

The retiming algorithm retimes only edge-triggered registers. It does not retime level-sensitive latches. Currently you can use retiming only for certain Actel, Altera, and Xilinx families. The option is not available if it does not apply to the family you are using.

These sections contain detailed information about using retiming.

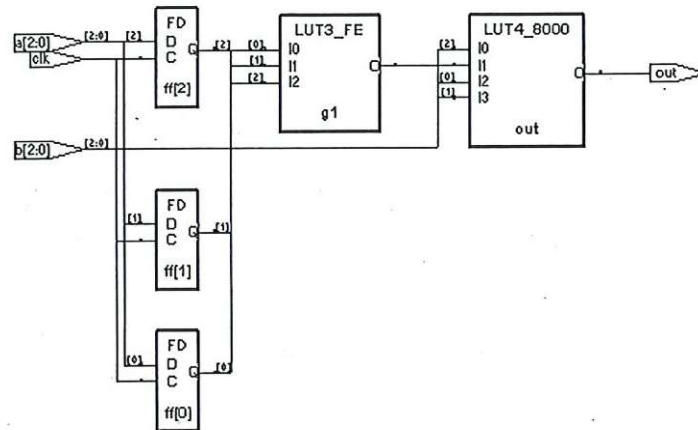
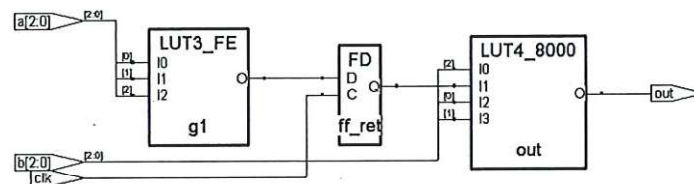
- Controlling Retiming, next
- Retiming Example, on page 6-46
- Retiming Report, on page 6-48
- How Retiming Works, on page 6-48

Controlling Retiming

The following procedure shows you how to use retiming.

1. To enable retiming for the whole design, check the Retiming check box.

You can set the Retiming option from the button panel in the Project window, or with the Project->Implementation Options command (Device tab). The option is only available in certain technologies.

☐ Retiming

☒ Retiming
☒ Pipelining


The top figure shows two levels of logic between the registers and the output, and no levels of logic between the inputs and the registers.

The bottom figure shows the results of retiming the three registers at the input of the OR gate. The levels of logic from the register to the output are reduced from two to one. The retimed circuit has better performance than the original circuit. Timing is improved by transferring one level of logic from the critical part of the path (register to output) to the non-critical part (input to register).

Pipelining

- **Definition:** Inserting extra registers into a long combinational path.
- **Goal:** Reduce the critical path delay so the design can run at a higher clock frequency.
- **Effect on performance:**
 - **Throughput:** Increases (more results per unit time).
 - **Latency:** Increases (more cycles before a result emerges).
- **Example:** Splitting a multiplier-adder chain into two stages with a register in between.
- **FPGA impact:** Uses more flip-flops, but often essential for high-speed DSP or arithmetic designs [All About Cir... +1](#) .

Retiming

- **Definition:** Moving existing registers forward or backward across combinational logic.
- **Goal:** Balance the delay across pipeline stages without changing the number of registers.
- **Effect on performance:**
 - **Throughput:** Can improve by shortening the longest path.
 - **Latency:** Stays the same (no new registers added).
- **Example:** If one stage has a very short path and the next stage has a very long path, retiming shifts registers so both stages have balanced delays.
- **FPGA impact:** Supported by synthesis tools (e.g., Xilinx Vivado), but effectiveness is limited compared to ASIC tools [Reddit +1](#) .

Inferring RAMs

There are two methods of handling RAMs: instantiation and inference. The software can automatically infer RAMs if they are structured correctly in your source code. For details, see the following sections:

- Inference vs. Instantiation, next
- Coding RAMs for Inference, on page 6-55
- Specifying RAM Implementation Styles, on page 6-59
- Implementing Altera RAMs Automatically, on page 6-61
- Implementing Xilinx RAMs Automatically, on page 6-64
- Implementing Altera RAMs: FLEX and APEX, on page 6-67
- Implementing Altera RAMs: Stratix Multi-Port RAMs, on page 6-69
- Inferring Xilinx Block RAMs Using Registered Addresses, on page 6-70
- Inferring Xilinx Block RAMs Using Registered Output, on page 6-73
- Setting Xilinx RAM Initialization Values, on page 6-78
- Mapping Xilinx ROM to Block RAM, on page 6-79

Inference vs. Instantiation

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

Inference in Synthesis	Instantiation
Advantages	Advantages
Portable coding style	Most efficient use of the RAM primitives of a specific technology
Automatic timing-driven synthesis	Supports all kinds of RAMs
No additional tool dependencies	

The mapper maps the RAM to ALTSYNCRAM in BIDIR_DUAL_PORT mode, which is fully synchronous. The actual ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

7. To implement Stratix multi-ports RAMs automatically, see *Implementing Altera RAMs: Stratix Multi-Port RAMs*, on page 6-69.

Implementing Xilinx RAMs Automatically

The following procedure shows you how to implement various Xilinx RAMs automatically. You can always override the automatic implementation by specifying the `syn_ramstyle` attribute, as described in *Specifying RAM Implementation Styles*, on page 6-59.

1. Follow the guidelines described for RAM inference by the compiler (*Coding RAMs for Inference*, on page 6-55).

The Xilinx mapper does not implement any RAMs that are not first inferred by the compiler.

2. To automatically implement distributed RAM, do the following:
 - Make sure the RAM size is at least 2K.
 - Make sure the write operation is synchronous and the read operation is asynchronous.

The Xilinx mapper implements RAMs inferred by the compiler as asynchronous RAMs, using the CLB resources.

3. To implement block SelectRAM+, do the following:
 - Make the write port synchronous. The read port can be asynchronous.
 - Register the read address (see *Inferring Xilinx Block RAMs Using Registered Addresses*, on page 6-70).
 - Make sure the RAM is a minimum of 2K bits.

The Xilinx mapper automatically implements RAMs inferred by the compiler as Block SelectRAM+, using the dedicated memory resources on the FPGA. The enable pin is tied to active and the reset pin is tied to inactive.

4. To implement single-port block RAM automatically, do the following:

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software.

Mapping Xilinx ROM to Block RAM

For Xilinx Virtex architectures, the software can map ROM into block RAM, provided you follow the guidelines in this procedure.

1. Place a dff register in front of the ROM, or place one of the following after the ROM:

Asynchronous	Synchronous
dff, dffe	
dffr, dffre	sdffr, sdffre
dffs, dffse	sdffs, sdffse
dffpatr, dffpatre	sdffpatr, sdffpatre

where dffe is an enabled flip-flop, dffre is an enabled flip-flop with asynchronous reset, dffse is an enabled flip-flop with asynchronous set, and dffpatre is an enabled, vectored flip-flop with asynchronous reset pattern.

2. Ensure that the registers and ROMs are within the same hierarchy.
3. Ensure that the number of outputs of the candidate ROM is 64 or fewer.
4. Make sure that at least half the addresses possess assigned values. For example, in a ROM with ten address bits (1024 unique addresses), at least 512 of those unique addresses must be assigned values.
5. Specify the `syn_romstyle` attribute with the value set to `block_rom`.
6. Synthesize the design.

The software maps the ROM into block RAM.

Working with Gated Clocks

The gated clock feature is only available in the Synplify Pro and Synplify Premier tools. This section first describes the gated clock solution, which is available for certain Altera and Xilinx technology families. The information is organized into these sections:

- The Synplicity Approach to Gated Clocks, next
- Synthesizing a Gated Clock Design, on page 6-101
- Prerequisites for Gated Clock Conversion, on page 6-103
- Gated Clock Conversion Report, on page 6-105
- Gated Clocks for Black Boxes, on page 6-108
- Restrictions to Using Fix Gated Clocks, on page 6-110
- Generated-Clock Optimization, on page 6-111

The Synplicity Approach to Gated Clocks

ASIC designs typically gate clocks to conserve power, with custom clock trees defined for each individual tree. FPGA design has dedicated resources for low-skew clock nets. If an FPGA design implements a large number of customized clock trees on some other routing resource, it can result in clock skew and timing problems.

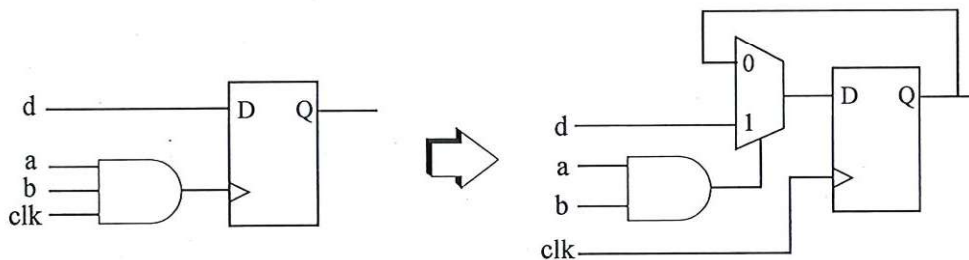
If you use the dedicated FPGA global clock trees instead, you free up routing resources and expedite placement and routing. Dedicated FPGA clock trees are routed to every sequential device on the die and are designed with low skew to avoid hold-time violations. Using these global clock trees allows the programmable routing resources of the FPGA to be used primarily for logic interconnect and simplifies static timing analysis because checks for hold-time violations based on minimum delays are unnecessary.

The solution is to separate the gating from the clock inputs, and combine individual clocks trees on the dedicated FPGA global clock trees. The software logically separates the gating from the clock and routes the gating to the clock enables on the sequential devices, using the programmable routing resources of the FPGA. The software separates a clock net going through an AND, NAND, OR, or NOR gate by doing one of the following:

- Inserting a multiplexer in front of the input pin of the synchronous element and connecting the clock net directly to the clock pin
- Moving the gating from the clock input pin to the dedicated enable pin, when this pin is available.

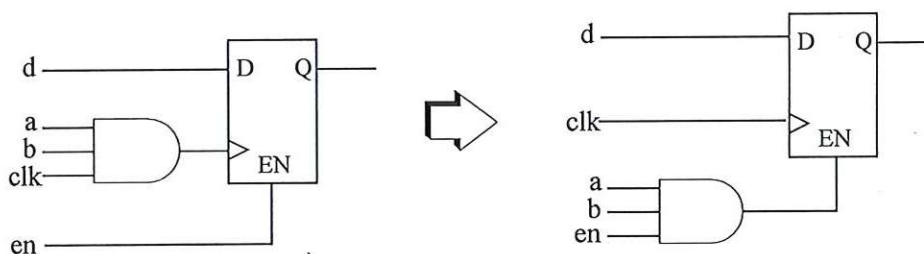
The ungated or base clock is routed to the clock inputs of the sequential devices using the global FPGA clock resources. Typically, many gated clocks are derived from the same base clock, so separating the gating from the clock allows a single global clock tree to be used for all gated clocks that reference that base clock.

See the following figure for examples of eliminating gated clocks.



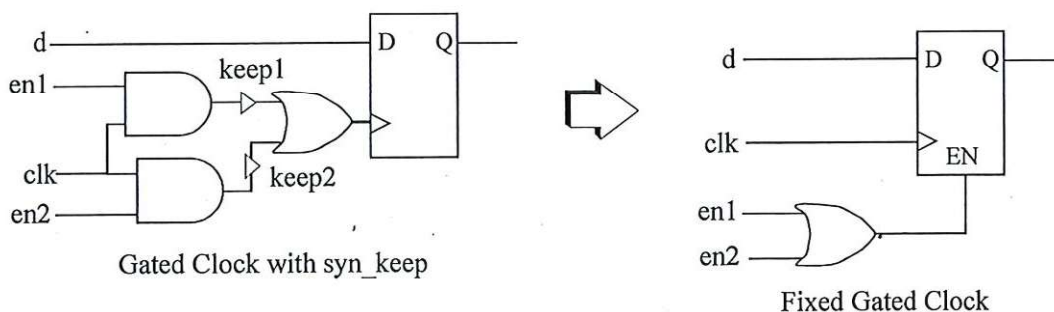
Gated Clock

Fixed Gated Clock



Gated Clock

Fixed Gated Clock



Gated Clock with syn_keep

Fixed Gated Clock

CHAPTER 7

Design Planning and Optimizations

Using the Synplify Premier Design Planner tool, you create a design plan to physically constrain portions of a design to specific regions on a device. Netlist restructure files usually contain primitives that have been bit sliced or modules that have been zippered. The design plan (.sfp) as well as netlist restructure files are used during optimization to improve the overall design performance. It is important to place physical constraints carefully. This section presents the following topics:

- Using the Design Planner on page 7-2
- Pin Assignments on page 7-6
- Working with Regions on page 7-19
- Checking Synplify Premier Utilization on page 7-25
- Using Process-Level Hierarchy on page 7-25
- Bit Slicing on page 7-26
- Zippering on page 7-33

You should be familiar with the recommended Synplify Premier design flows and the Design Planner tool to use these design planning tips effectively.

2. Select the destination region and paste.

Pin Assignments

This section discusses the following general guidelines for displaying and assigning pin assignments for design planning:

- Methods for Specifying Pin Assignments on page 7-6
- Specifying Pins Using the Design Plan Editor on page 7-7
- Implementing Pin Assignments on page 7-11
- Storing Temporary Pin Assignments on page 7-14
- Displaying Rats Nesting on page 7-15
- Assigning Clock Pins on page 7-17

Methods for Specifying Pin Assignments

You can specify I/O pin locks and pin assignments using any one of the following features of the Synplify Premier tool:

- Add pins to a constraint (.sdc) file using the SCOPE editor.
- Convert locked I/O pins to a constraint (.sdc) file using Run->Translate Constraints in the Project view. See *Converting Pin Location Constraint Files in the Synplify Premier Tool* on page 3-43 for details.
- Using the Design Plan Editor in Design Planner to assign pins. See *Specifying Pins Using the Design Plan Editor* on page 7-7 for details.
- Importing pin assignments from an Altera .pin or a Xilinx .pad constraint file to a design plan (.sfp) file using Tools->Import Info in the Design Planner. See *Importing Pin Location Files* on page 7-10.

CHAPTER 8

Vendor-Specific Optimizations

This chapter covers techniques for optimizing your design for various vendors. The information in this chapter is intended to be used together with the information in Chapter 6, *Design Optimization*.

This chapter describes the following:

- Passing Information to the P&R Tools, on page 8-2
- Generating Vendor-Specific Output, on page 8-6
- Working with Actel Designs, on page 8-8
- Working with Altera Designs, on page 8-11
- Working with Lattice Designs, on page 8-23
- Working with Xilinx Designs, on page 8-28

5. Start the Lattice ispLEVER place-and-route tool and run the Map stage. The place-and-route software includes the constraints from the synthesis .prf file when it generates another .prf file after mapping.
6. Run the PAR and BIT stages in ispLEVER.

Working with Xilinx Designs

This section contains tips for working with Xilinx designs:

- Designing for Xilinx Architectures, next
- Instantiating CoreGen Cores, on page 8-29
- Packing Registers for I/Os, on page 8-33
- Controlling Placement with RLOCs, on page 8-35
- Using Clock Buffers in Virtex Designs, on page 8-36
- Reoptimizing With EDIF Files, on page 8-39
- Instantiating Special I/O Standard Buffers for Virtex, on page 8-38

For additional Xilinx-specific techniques, see *The Xilinx MultiPoint Synthesis Flow*, on page 10-48, *Using the Xilinx Modular Flow*, on page 10-54, *Working with Gated Clocks*, on page 6-99, *Inferring RAMs*, on page 6-54, and *Inferring Shift Registers*, on page 6-80. Note that some of these features are not available in the Synplify product.

Designing for Xilinx Architectures

The tips listed here are in addition to the technology-independent design tips described in *Design Guidelines*, on page 6-2.

- For critical paths, attach the `xc_fast` attribute to the I/Os.
- To ensure that frequency constraints from register to output pads are forward annotated to the P&R tools, add default `input_delay` and `output_delay` constraints of 0.0 in the synthesis tool. The synthesis tool forward-annotates the frequency constraints as PERIOD constraints

(register-to-register) and OFFSET constraints (input-to-register and register-to-output). The place-and-route tools use these constraints.

- Run successive place-and-route iterations with progressively tighter timing constraints to get the best results possible.
- Specify a UNISIM library using the following syntax:

```
library unisim;
use unisim.vcomponents.all;
```

Remove any other package files with user-defined UNISIM primitives.

Instantiating CoreGen Cores

Predesigned IP cores save on design effort and improve performance. The process for handling IP cores is slightly different for CoreGen and Virtex PCI cores. The following procedure describes how to instantiate a CoreGen module. For Virtex PCI cores, see *Instantiating Virtex PCI Cores*, on page 8-30.

1. Define the core as a black box by adding the `syn_black_box` attribute to the module definition line.

```
module ram64x8(din, addr, we, clk, dout)/* synthesis syn_black_box
*/;
    input [7:0] din;
    input [5:0] addr;
    input we, clk;
    output [7:0] dout;
endmodule;
```

2. Make sure the bus format matches the bus format in the core generator, using the `syn_edif_bit_format` and `syn_edif_scalar_format` directives if needed.

```
module ram64x8(din, addr, we, clk, dout)
/* synthesis syn_black_box syn_edif_bit_format = "%u<%i>"
syn_edif_scalar format = "%u" */;
```

3. Generate timing and resource usage information for synthesis.
 - Use the Xilinx CORE generator to create structural EDIF netlists. For legacy cores, generate a single flat .edf netlist file. For newer cores, generate a top-level flat .edf netlist file that instantiates .ndf files for each hierarchical level in the design.
 - In the synthesis software, add the generated files (.edf only for legacy cores; .edf and .ndf for newer cores) to your project.

4. Instantiate the black box in the module or architecture.

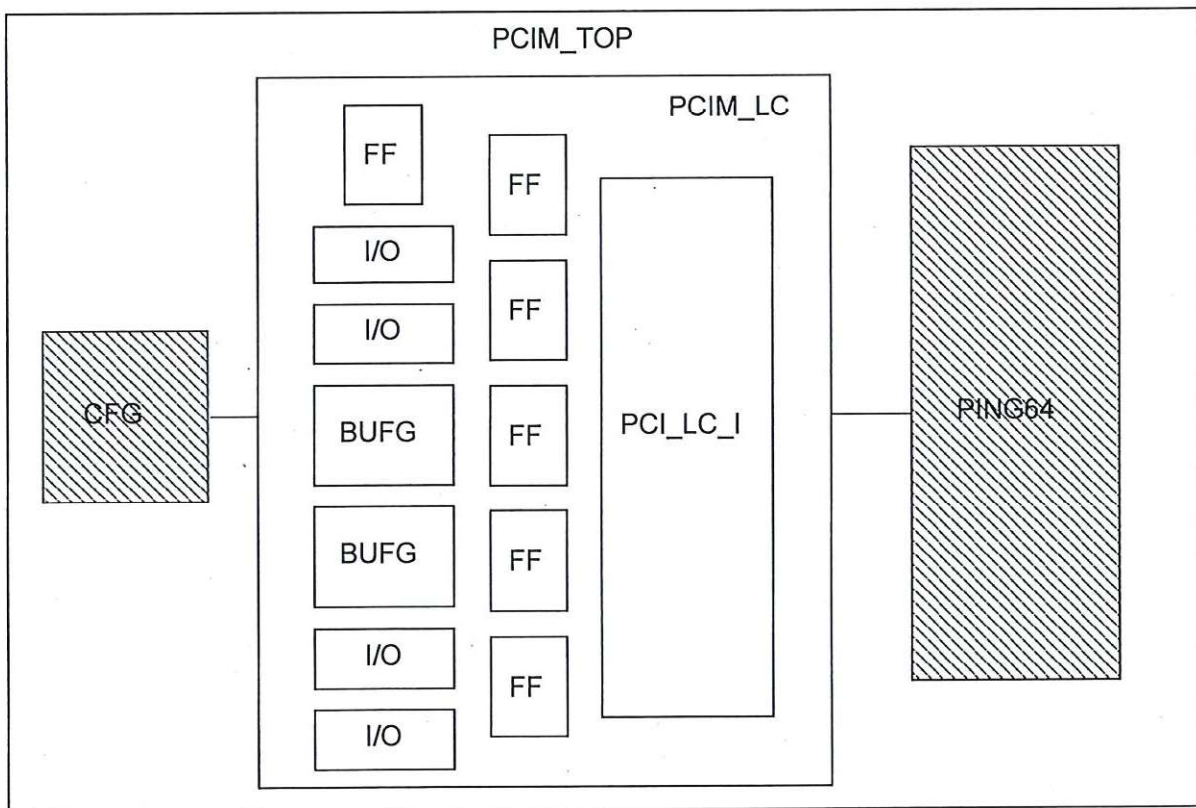
```
ram64x8 r1(din, addr, we, clk, dout);
```

5. Synthesize the design.

If you supplied structural EDIF netlists, the software optimizes the design based on the information in the structural netlists. The generated reports contain the optimization information .

Instantiating Virtex PCI Cores

For Virtex PCI cores, you can use either a top-down or bottom-up methodology. This figure shows a design that is used in the explanations of both methodologies, below.



Working with Tcl Scripts and Commands

The software uses extensions to the popular Tcl (Tool Command Language) scripting language to control synthesis and for constraint files. See the following for more information:

- Crossprobing from the Tcl Script Window, next
- Using Tcl Commands and Scripts, on page 10-4
- Generating a Job Script, on page 10-5
- Creating a Tcl Synthesis Script, on page 10-5
- Using Tcl Variables to Try Different Clock Frequencies, on page 10-7
- Using Tcl Variables to Try Several Target Technologies
- Running Bottom-up Synthesis with a Script, on page 10-9

You can also use synhooks Tcl scripts, as described in *Automating Flows with synhooks.tcl*, on page 10-10.

Crossprobing from the Tcl Script Window

To crossprobe from the Tcl Script window (not a Synplify feature) to the source code, click on the Warnings tab and then double-click a line in the Tcl window. The software opens the relevant source code file and highlights the corresponding code. Crossprobing from the Tcl script window is useful for debugging error messages.

Using Tcl Commands and Scripts

1. To get help on Tcl syntax, do any of the following:
 - Refer to the online help (Help -> Tcl Help) for general information about Tcl syntax.
 - Refer to the *Reference Manual* for information about the synthesis commands.
 - Type `help *` in the Tcl window for a list of all the Tcl synthesis commands. The Tcl window is not available in Synplify.

- Type `help <command_keyword>` in the Tcl window to see the syntax for the command.
2. To run a Tcl script, do the following:
 - Create a Tcl script. Refer to *Generating a Job Script*, on page 10-5 and *Creating a Tcl Synthesis Script*, on page 10-5.
 - Run the Tcl script by either typing `source Tcl_scriptfile` in the Tcl script window, or selecting File -> Run Tcl Script, selecting the Tcl file and clicking Open.

The software runs the selected script and executes the commands in it. For more information about Tcl scripts, refer to the following sections.

Generating a Job Script

You can record Tcl commands from the interface and use it to generate job scripts.

1. In the Tcl script window, type recording `-file logfile` to write out a Tcl log file.
2. Work through a synthesis session.

The software saves the commands from this session into a Tcl file that you can use as a job script, or as a starting point for creating other Tcl files.

Creating a Tcl Synthesis Script

Tcl scripts are text files with a *.tcl extension. You can use the graphic user interface to help you create a Tcl script. Interactive commands that you use actually execute Tcl commands, which are displayed in the Tcl window as they are run. You can copy this text in the Tcl window and paste it into a text file that you build to run as a Tcl script. For example:

```
add_file -verilog "prep2.v"
set_option -technology STRATIX
set_option -part EP1SGX40D
set_option -package FC1020
```

```
project -run
```

```
open_file -rtl_view
open_file -technology_view
```

6. Check the syntax.

- Check case, because Tcl is case-sensitive.
- Start all comments with a hash mark (#).
- Enclose all pathnames and filenames in double quotes.
- Always use a forward slash (/) in directory and pathnames, even on the PC.

Using Tcl Variables to Try Different Clock Frequencies

To create a single script for multiple synthesis runs with different clock frequencies, you need to create a Tcl variable for the different settings you want to try. For example, you might want to try different target technologies.

1. To create a variable, use this syntax:

```
set variable_name {
    first_option_to_try
    second_option_to_try
    ...}
```

2. Create a foreach loop that runs through each option in the list, using the appropriate Tcl commands. The following example shows a variable set up to synthesize a design with different frequencies. It also creates a separate log file for each run.

```
Set of frequencies to try → set try_freq {
    85.0
    90.0
    92.0
    95.0
    97.0
    100.0
}

Foreach loop → foreach frequency $try_freq {
    set_option -frequency $frequency
    project -log_file $frequency.srr
    project -run}
```

Tcl commands that set the frequency, create separate log files for each run, and run synthesis

```

/*
** This example shows the use of the Vivado use_dsp48 attribute
**
** Acceptable values are:
** "yes" - Allows user to manually force implementation of arithmetic
logic into DSP blocks
** "no" - Allows user to manually prevent implementatino of logic on
DSP blocks
**
** This attribute can be placed in the RTL on signals, architectures
and components, entities and modules
**
** The default behavior is for Vivado Synthesis to determine the
correct behavior for the logic
**
*****
*****/

(* use_dsp48 = "yes" *)
module use_dsp48_example (clk, A, B, Q);

input clk;
input [7:0] A;
input [3:0] B;
output reg [11:0] Q;

always @(posedge clk)
begin

Q <= A * B;

end

endmodule

```



```

/*
** This example shows the use of the Vivado use_dsp48 attribute
**
** Acceptable values are:
** "yes" - Allows user to manually force implementation of arithmetic
logic into DSP blocks
** "no" - Allows user to manually prevent implementation of logic on
DSP blocks
**
** This attribute can be placed in the RTL on signals, architectures
and components, entities and modules
**
** The default behavior is for Vivado Synthesis to determine the
correct behavior for the logic
**
*****
*****/

module use_dsp48_example(a, q, clk, rst);

output [7:0] q;
input clk;
input rst;
input [7:0] a;

(* use_dsp48 = "yes" *)
reg [7:0] s;

always @(posedge clk)
begin
    if (rst)
        begin
            s <= 'b0;
        end
    else
        begin
            s <= s + a;
        end
end //always block

assign q = s;

endmodule

```

Synchronous FIFO Homework Problem

1 Preliminaries

Two computers must transfer data (see figure 1).

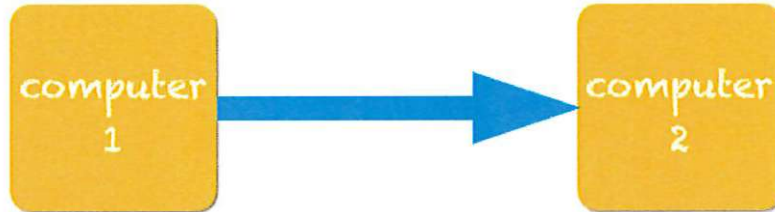


Figure 1:

Computer 1 sends the data at a rate computer 2 cannot accept. Figure 2 shows one possible solution: insert a first-in-first-out (FIFO) buffer between the two computers.

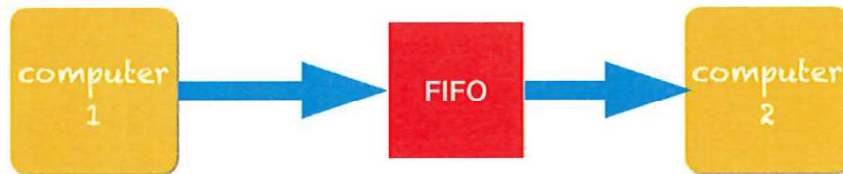


Figure 2:

The FIFO is a buffer with a depth and a width. The depth is the number of buffer (memory) locations and the width is the number of bits per location. The first data entering the FIFO buffer is the first data read from the FIFO buffer. (Computer 1 writes data into the FIFO; computer 2 reads data from the buffer.)

Figure 3 shows a block diagram of the FIFO. The data input (din) in bits is set by the FIFO width. The data output (dout) is the same width. . This

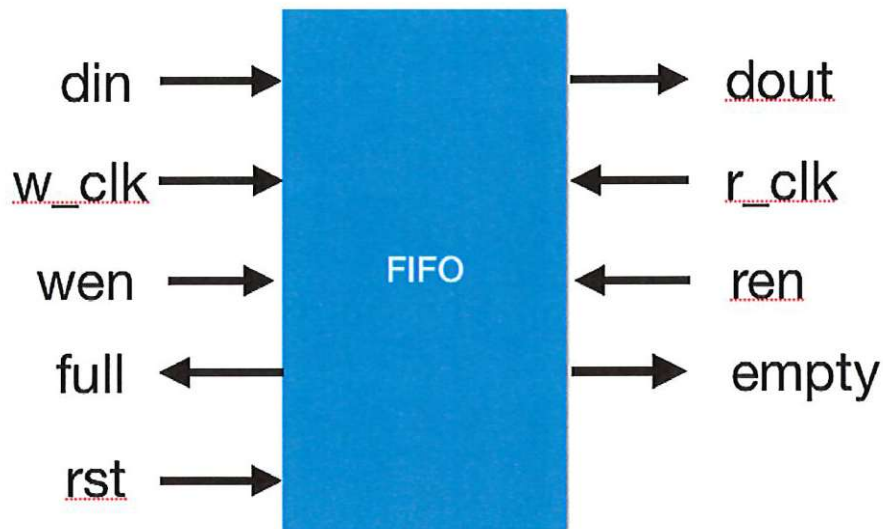


Figure 3:

FIFO is synchronous so the write clock (w_clk) and the read clock (r_clk) are the same clock signal. Data is put into the FIFO synchronously whenever the write enable (wen) is asserted. Similarly data is read synchronously from the FIFO whenever read enable (ren) is asserted. Clocks are positive edge triggered. The read and write enables are active high.

The FIFO depth equals the number buffer locations. Internally there are two k -bit pointers (w_ptr and r_ptr) where $k = \log_2(\text{depth})$. These pointers specify which buffer location is written to or read from. That is, w_ptr points to the location where the next data input will be stored while r_ptr which location data will be read from. The concept is shown in figure 4. After a write operation the w_ptr increments and the r_ptr increments after every read operation. This is a circular buffer.

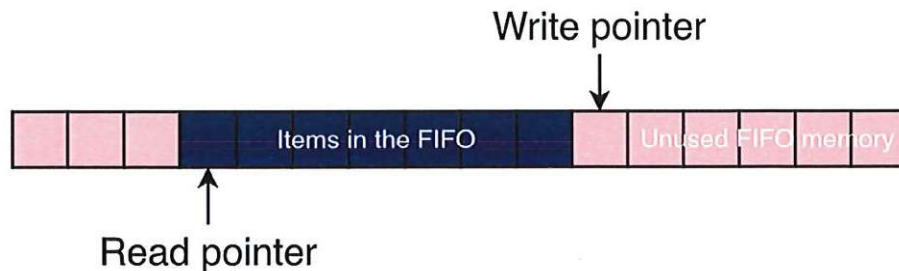


Figure 4:

Of course you should never write to a full FIFO or read from an empty FIFO. Two flags (status signals) are used. 'empty' indicates there is no data in the FIFO. This flag is read by computer 2 before starting a read operation. 'full' indicates there are no available buffer locations left to store data. Computer 1 should check this flag before starting a write operation. Both flags are active high.

The empty flag is asserted if $w_ptr == r_ptr$. (You need to determine when the full flag should be asserted.) Both flags are asynchronous.

2 Assignment

This homework assignment is worth 30 points.

You are to write a Verilog description of a synchronous FIFO with data width= 8. The FIFO has depth= 8.

You are also required to write a test bench to simulate your design. The test bench should perform the following sequence of operations:

1. reset the FIFO and perform any required initialization
2. perform two write operations
3. perform one read operation
4. perform two write operations
5. perform two read operations
6. perform as many write operations as necessary to fill the FIFO
7. perform one read operation

The first write operation should store 00H, the second 55H, the third AAH and the fourth FFH. Repeat this sequence for any future write operations.

Now simulate your design using your testbench. Take a screenshot of the timing diagram produced. The timing diagram should show

- the reset signal
- the clock
- both read and write pointers
- both read and write enables
- data inputs
- data outputs
- the empty flag
- the full flag

Upload the timing diagram, the HDL source code and the testbench code to canvas.

3 Specific Requirements

Your testbench must perform the following four actions:

1. initialize the FIFO
2. create a clock signal
3. terminate the simulation after a fixed period time
4. perform the sequence of read/write operations given in the previous section.

Use initial and/or always blocks to execute action 4.

Actions 1, 2 and 3 above **MUST** be coded as a Verilog task. Put each action in a separate, independent task. (Three actions, therefore you need to code three separate tasks.) Construct a 4th task called **SIM** that calls these three tasks.

THE THREE SUB-TASKS IN SIM MUST EXECUTE SIMULTANEOUSLY.