

Low Power in Encounter® RTL Compiler

**Product Version 7.2
December 2007**

© 2003-2007 Cadence Design Systems, Inc. All rights reserved.
Portions © Concept Engineering GmbH. Used by permission.
Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Product Encounter™ RTL Compiler contains technology licensed from, and copyrighted by: Concept Engineering GmbH, and is ©1998-2006, Concept Engineering GmbH. All rights reserved.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Patents: Cadence Product Encounter™ RTL Compiler described in this document, is protected by U.S. Patents [5,892,687]; [6,470,486]; 6,772,398; [6,772,399]; [6,807,651]; [6,832,357]; and [7,007,247]

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

<u>List of Figures</u>	13
------------------------------	----

<u>List of Examples</u>	17
-------------------------------	----

<u>Preface</u>	19
----------------------	----

<u>About This Manual</u>	20
--------------------------------	----

<u>Additional References</u>	20
------------------------------------	----

<u>How to Use the Documentation Set</u>	21
---	----

<u>Reporting Problems or Errors in Manuals</u>	22
--	----

<u>Customer Support</u>	22
-------------------------------	----

<u>SourceLink Online Customer Support</u>	22
---	----

<u>Other Support Offerings</u>	22
--------------------------------------	----

<u>Messages</u>	23
-----------------------	----

<u>Man Pages</u>	23
------------------------	----

<u>Command-Line Help</u>	24
--------------------------------	----

<u>Getting the Syntax for a Command</u>	24
---	----

<u>Getting the Syntax for an Attribute</u>	24
--	----

<u>Searching for Attributes</u>	25
---------------------------------------	----

<u>Searching For Commands When You Are Unsure of the Name</u>	25
---	----

<u>Documentation Conventions</u>	26
--	----

1

<u>Introduction</u>	27
---------------------------	----

<u>Today's Power Challenges</u>	28
---------------------------------------	----

<u>Traditional Low Power Synthesis Features</u>	28
---	----

<u>Power Analysis</u>	29
-----------------------------	----

<u>Power Optimization</u>	29
---------------------------------	----

<u>Support for Advanced Power Management Techniques through the Common Power Format</u>	33
--	----

<u>Required Files for Low Power Synthesis</u>	34
---	----

2

<u>Recommended Flow</u>	37
<u>Synthesis Flow with Low Power Features</u>	38
<u>Flow Steps</u>	39
<u>Begin Setup</u>	39
<u>Read Target Libraries</u>	40
<u>Enable Clock Gating, Operand Isolation, RTL Power Analysis</u>	40
<u>Read HDL Files, Elaborate Design, and Insert Operand Isolation</u>	40
<u>Apply Clock-Gating Directives</u>	41
<u>Set Timing and Design Constraints</u>	41
<u>Apply Optimization Directives</u>	42
<u>Set Leakage Power and Dynamic Power Constraints</u>	42
<u>Annotate Switching Activities</u>	43
<u>Synthesize Design</u>	43
<u>Annotate Switching Activities</u>	44
<u>Analyze Power</u>	44
<u>Analyze Design</u>	44
<u>Export to Place and Route</u>	45

3

<u>Power Analysis Concepts</u>	47
<u>Power Dissipation Components</u>	48
<u>Leakage Power Dissipation</u>	48
<u>Net Power Dissipation</u>	48
<u>Internal Power Dissipation</u>	49
<u>How the RC-LP Engine Models Power Dissipation</u>	50
<u>Power Calculation</u>	51
<u>Internal Cell Power Calculation</u>	52
<u>Net Power Calculation</u>	54
<u>Leakage Power Calculation</u>	54

4

<u>Providing Switching Activity Information</u>	57
<u>Introduction</u>	58

Low Power in Encounter RTL Compiler

<u>Sources of Switching Activity Information</u>	60
<u>Using Default Switching Activities</u>	61
<u>Propagating Net Switching Activities</u>	62
<u>Overriding Net Switching Activities</u>	62
<u>Determining the Source of the Switching Activity Information</u>	63
<u>Reading Switching Activity Information from a TCF File</u>	64
<u>Reading a TCF File with Modified Clock Frequency</u>	65
<u>Reading Multiple TCF Files with Different Weights</u>	66
<u>Reading Instance-Based Switching Activities</u>	67
<u>Reading Switching Activity Information from an SAIF File</u>	71
<u>Reading Switching Activity Information from a VCD File</u>	72
<u>Reading Instance-Based Switching Activities</u>	73
<u>Displaying an Activity Profile</u>	75
<u>Checking System Messages when Reading Switching Activities</u>	80

5

<u>Power Analysis</u>	83
<u>Introduction</u>	84
<u>RTL Power Analysis</u>	86
<u>Reporting Clock Tree Power</u>	89
<u>Reporting on All Power Components</u>	92
<u>Reporting Power for the Design</u>	92
<u>Limiting the Number of Hierarchy Levels Shown</u>	93
<u>Reporting the Power of the Leaf Instances</u>	93
<u>Reporting Power for Specific Instances</u>	94
<u>Sorting According to One Power Component</u>	94
<u>Net-based Power Reports</u>	96
<u>Showing User-Asserted Information</u>	96
<u>Reporting Power on Leaf Instances</u>	97
<u>Reporting Power Consumption per Used Cell Types</u>	98
<u>Reporting on Specific Power Components for Specific Objects</u>	99
<u>Getting Power-Related Information on the Design</u>	99
<u>Getting Power-Related Information on an Instance</u>	100
<u>Getting Power-Related Information on a Pin, Port, or Subport</u>	100
<u>Getting Power-Related Information on a Net</u>	100

Low Power in Encounter RTL Compiler

Troubleshooting	102
<u>Common Causes of Correlation Problems</u>	102
<u>Debugging Techniques To Find Cause of Correlation Problem</u>	104
<u>Changing Clock Frequency Does not Have Expected Result</u>	104
6	
<u>Clock Gating</u>	105
Introduction	106
RTL Coding Style Examples	109
Enabling Clock Gating	114
Controlling Handling of Timing Exceptions during Clock Gating	114
Controlling Naming of Modules, Nets, and Ports	114
Controlling Selection of Clock-Gating Logic	115
Specifying the Library Cell Name of the Integrated Clock-Gating Cell	117
Defining a Clock-Gating Module	117
Using Attributes to Select an Integrated Clock-Gating Cell in the Technology Library ..	122
Controlling Insertion of Clock-Gating Logic	130
Previewing the Effect of Clock Gating	133
Clock Gating with DFT	135
Selecting Clock-Gating Logic with Control Logic	135
Specifying the Control Signals	136
Connecting the Test Signals	137
Selecting Clock-Gating Logic with Observability Logic	139
Inserting the Observability Logic	139
Recommended Top-Down Clock Gating Flow with DFT	141
Checking System Messages during Optimization	144
Reporting Clock-Gating Information	146
Clock-Gating Post-Processing	152
Decloning Clock-Gating Instances	152
Removing Clock-Gating Instances	156
Multi-Stage Clock Gating	159
Creating Shared Clock-Gating Logic Using Common Enable	159
Splitting Clock-Gating Instances into Multiple Levels of Clock-Gating Logic	164
Consolidating Multi-Stage Clock-Gating Logic	169

Low Power in Encounter RTL Compiler

<u>Inserting Clock Gating in a Mapped Netlist</u>	170
<u>Troubleshooting</u>	173
<u>Limitations</u>	174

7

<u>Operand Isolation</u>	175
<u>Introduction</u>	176
<u>RTL Coding Style Examples</u>	179
<u>Enabling Operand Isolation</u>	180
<u>Controlling Naming of Modules, Nets, and Ports</u>	180
<u>Finding Operand-Isolation Logic in the Design Hierarchy</u>	180
<u>Checking System Messages during Optimization</u>	181
<u>Reporting Operand Isolation Information</u>	182
<u>Limitations</u>	185

8

<u>Leakage Power Optimization</u>	187
<u>Introduction</u>	188
<u>Prerequisites</u>	191
<u>Enabling Leakage Power Optimization</u>	192
<u>Selecting Leakage Power Optimization Effort when Using Multiple Vth Libraries</u>	193
<u>Controlling Power Optimization</u>	194
<u>Reducing Dynamic Power First</u>	194
<u>Using Weight Factors</u>	194
<u>Preventing Leakage Power Optimization</u>	195
<u>Checking System Messages during Optimization</u>	196
<u>Reporting Leakage Power</u>	198

9

<u>Dynamic Power Optimization</u>	199
<u>Introduction</u>	200
<u>Prerequisites</u>	203
<u>Enabling Dynamic Power Optimization</u>	204
<u>Controlling Power Optimization</u>	205

Low Power in Encounter RTL Compiler

<u>Preventing Dynamic Power Optimization</u>	205
<u>Checking System Messages during Optimization</u>	206
<u>Reporting Dynamic Power</u>	207
<u>Troubleshooting</u>	209
<u>Clock Gating Yields Negligible Dynamic Power Reduction</u>	209

10

<u>Using CPF for Multiple Supply Voltage Designs</u>	211
<u>Overview</u>	212
<u>CPF File for MSV Design</u>	213
<u>MSV Information in the Design Information Hierarchy</u>	215
<u>Flow Steps</u>	217
<u>Read Target Libraries</u>	219
<u>Read CPF File</u>	220
<u>Check the CPF File</u>	221
<u>Set Timing, Design and Power Constraints</u>	222
<u>Apply Clock-Gating and Optimization Directives</u>	222
<u>Annotate Switching Activities</u>	223
<u>Estimate RTL Power</u>	223
<u>Synthesize Design</u>	223
<u>Reload CPF File</u>	224
<u>Execute the CPF File</u>	224
<u>Verify Added Power Logic</u>	225
<u>Run Incremental Optimization</u>	226
<u>Annotate switching activities</u>	226
<u>Analyze Power</u>	226
<u>Analyze Design</u>	227
<u>Export to Place and Route</u>	230

11

<u>Using CPF for Designs Using Power Shutoff Methodology</u>	233
<u>Overview</u>	234
<u>CPF File for PSO Design</u>	236
<u>PSO Information in the Design Information Hierarchy</u>	239
<u>Flow Steps</u>	242

Low Power in Encounter RTL Compiler

<u>Read Target Libraries</u>	244
<u>Read CPF File</u>	245
<u>Check the CPF File</u>	246
<u>Set Timing, Design and Power Constraints</u>	246
<u>Apply Clock-Gating and Optimization Directives</u>	247
<u>Annotate Switching Activities</u>	247
<u>Estimate RTL Power</u>	248
<u>Synthesize Design</u>	249
<u>Reload CPF File</u>	250
<u>Execute CPF File</u>	250
<u>Verify Added Power Logic</u>	254
<u>Run Incremental Optimization</u>	254
<u>Annotate switching activities</u>	254
<u>Analyze Power</u>	254
<u>Analyze Design</u>	256
<u>Export to Place and Route</u>	256

12

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

<u>Overview</u>	260
<u>CPF File for DVFS Design</u>	264
<u>Additional Information in the Design Information Hierarchy</u>	269
<u>Flow Steps</u>	270
<u>Read Target Libraries</u>	272
<u>Read CPF File</u>	273
<u>Check the CPF File</u>	275
<u>Set Timing, Design and Power Constraints</u>	275
<u>Apply Clock-Gating and Optimization Directives</u>	276
<u>Annotate Switching Activities</u>	276
<u>Estimate RTL Power</u>	276
<u>Synthesize Design</u>	277
<u>Reload CPF File</u>	277
<u>Execute CPF File</u>	278
<u>Verify Added Power Logic</u>	282

Low Power in Encounter RTL Compiler

<u>Run Incremental Optimization</u>	282
<u>Annotate switching activities</u>	282
<u>Analyze Power</u>	282
<u>Analyze Design</u>	283
<u>Export to Place and Route</u>	284

13

<u>Other Flows</u>	287
<u>Recommended Bottom-Up Clock Gating Flow with DFT</u>	288
<u>Block-Level Synthesis</u>	289
<u>Top-Level Synthesis</u>	290
<u>Scan Insertion after Clock-Gating Insertion</u>	292
<u>Power Optimization when Starting with Mapped Netlist</u>	295
<u>State-Retention Cell Replacement when Starting with Mapped Netlist</u>	297

A

<u>CPF Support in RTL Compiler</u>	299
<u>Migrating from the Native Flow to the CPF Flow</u>	300
<u>Supported CPF Commands</u>	301
<u>Currently Not Supported Commands</u>	301
<u>Commands Not Applicable to Synthesis</u>	301
<u>Mapping of CPF Commands to RTL Compiler Commands and Attributes</u>	302
<u>CPF Version Specification</u>	302
<u>Library Specification</u>	302
<u>Library Cell-Related CPF Commands</u>	303
<u>Scope Commands</u>	304
<u>General Purpose Commands</u>	305
<u>Design Specification</u>	305

B

<u>Low Power Synthesis Reference</u>	311
<u>Commands</u>	312
<u>Attributes</u>	315

C

<u>Advanced Low Power Reference</u>	323
<u>Commands</u>	324
<u>Attributes</u>	325
<u>Index</u>	329

Low Power in Encounter RTL Compiler

List of Figures

Figure 1-1 Clock Gating Concept	30
Figure 1-2 Operand Isolation Concept	31
Figure 1-3 Data Flow for Low Power Synthesis in RTL Compiler Environment	34
Figure 2-1 Top-Down Low Power Synthesis Flow	38
Figure 3-1 Short Circuit Power Dissipation	49
Figure 3-2 Power Dissipation Model	50
Figure 4-1 Power Analysis in the Top-Down Low Power Synthesis Flow	59
Figure 4-2 Netlist of Full Design Loaded in RTL Compiler	67
Figure 4-3 Partial TCF File for Instance ma0	68
Figure 4-4 read tcf Summary	68
Figure 4-5 Netlist of Partial Design Loaded in RTL Compiler	69
Figure 4-6 Full Chip TCF File	70
Figure 4-7 read tcf Summary	70
Figure 4-8 Steps in Design Browser Window to Display Activity Profile	76
Figure 4-9 Activity Profile for the Design over Entire Simulation Time	77
Figure 4-10 Activity Profile for the Design over Specific Time Window	78
Figure 4-11 Activity Profile for the Design and Hierarchical Instance	78
Figure 5-1 Power Analysis in the Top-Down Low Power Synthesis Flow	85
Figure 5-2 Command Script for Clock Tree Power Estimation	90
Figure 6-1 Clock Gating Concept	106
Figure 6-2 Recommended Clock-Gating Flow Starting from RTL	107
Figure 6-3 Circuit with Flip-Flops without Inferred Synchronous Reset	110
Figure 6-4 Circuit with Flip-Flops without Inferred Synchronous Reset with Clock Gating	110
Figure 6-5 Circuit with Flip-Flops Inferred with a Synchronous Reset	111
Figure 6-6 Circuit with Flip-Flops Inferred with a Synchronous Reset with Clock Gating	111
Figure 6-7 Circuit with Flip-Flops Inferred with an Asynchronous Reset	112
Figure 6-8 Circuit with Flip-Flops Inferred with an Asynchronous Reset with Clock Gating	112

Low Power in Encounter RTL Compiler

112

<u>Figure 6-9 Hierarchical Clock Gating Example before Clock Gating</u>	113
<u>Figure 6-10 Hierarchical Clock Gating Example after Clock Gating</u>	113
<u>Figure 6-11 Selection of Clock-Gating Logic</u>	115
<u>Figure 6-12 Circuit with Negative Edge-Triggered Logic before and after Clock Gating</u>	119
<u>Figure 6-13 Module wrapper.v</u>	120
<u>Figure 6-14 Clock-Gated Design Using User-Created Clock-Gating Module</u>	121
<u>Figure 6-15 Flow Supporting User-Created Clock-Gating Module</u>	121
<u>Figure 6-16 Impact of a Glitch on a Latched Clock Gate (latch_posedge)</u>	124
<u>Figure 6-17 Impact of a Glitch on a Flip-Flop Gated Clock (ff_posedge)</u>	125
<u>Figure 6-18 Impact of a Glitch on a Non-Latched Clock Gate (none_posedge)</u>	126
<u>Figure 6-19 Netlist of Discrete Clock-gating Module</u>	129
<u>Figure 6-20 Clock Gating with Extracted Common Enable</u>	132
<u>Figure 6-21 Possible Locations of the Test-Control Logic</u>	136
<u>Figure 6-22 Clock-Gating Cells with Observability Logic</u>	139
<u>Figure 6-23 Example of Observability Logic</u>	140
<u>Figure 6-24 Merging Clock-Gating Instances</u>	152
<u>Figure 6-25 Clock Gating with and Without Sharing</u>	159
<u>Figure 6-26 Clock Gating before and after Splitting</u>	164
<u>Figure 6-27 Clock Gating before and after Consolidation</u>	169
<u>Figure 6-28 Clock-Gating Flow with DFT Starting from Mapped Netlist</u>	170
<u>Figure 7-1 Operand Isolation Concept</u>	176
<u>Figure 7-2 Recommended Operand Isolation Flow Starting from RTL</u>	177
<u>Figure 7-3 Partial Decommitment</u>	178
<u>Figure 8-1 Common Leakage Power Optimization Flow</u>	189
<u>Figure 9-1 Recommended Dynamic Power Optimization Flow</u>	201
<u>Figure 10-1 Example of MSV Design</u>	212
<u>Figure 10-2 CPF File for MSV Example</u>	213
<u>Figure 10-3 Design Information Hierarchy</u>	215
<u>Figure 10-4 Top-Down MSV Low Power Flow with CPF</u>	217
<u>Figure 10-5 Preserving Nets to Prevent Buffer Insertion</u>	225

Low Power in Encounter RTL Compiler

<u>Figure 10-6 Timing Report</u>	229
<u>Figure 10-7 Extract of a Sample Encounter Setup Template File</u>	230
<u>Figure 10-8 Sample Encounter Config Template File</u>	231
<u>Figure 11-1 Example of Design with PSO</u>	234
<u>Figure 11-2 CPF File for PSO Example</u>	236
<u>Figure 11-3 Design Information Hierarchy</u>	239
<u>Figure 11-4 Top-Down PSO Low Power Flow with CPF</u>	242
<u>Figure 11-5 Input and Output Subports Created when Connecting Enable Driver</u>	251
<u>Figure 11-6 Preserving Nets to Prevent Buffer Insertion</u>	253
<u>Figure 11-7 Extract of a Sample Encounter Setup Template File</u>	257
<u>Figure 11-8 Sample Encounter Config Template File</u>	258
<u>Figure 12-1 Example of DVFS Design</u>	261
<u>Figure 12-2 CPF File for DVFS Example</u>	264
<u>Figure 12-3 Design Information Hierarchy</u>	269
<u>Figure 12-4 Top-Down DVFS Low Power Flow with CPF</u>	270
<u>Figure 12-5 Input and Output Subports Created when Connecting Enable Driver</u>	278
<u>Figure 12-6 Preserving Nets to Prevent Buffer Insertion</u>	280
<u>Figure 12-7 Preserving Nets to Prevent Buffer Insertion</u>	281
<u>Figure 12-8 Extract of a Sample Encounter Setup Template File</u>	284
<u>Figure 12-9 Sample Encounter Config Template File</u>	285
<u>Figure 13-1 Example Design</u>	288
<u>Figure 13-2 Example Design</u>	292

Low Power in Encounter RTL Compiler

List of Examples

Example 3-1 Internal Power Calculation	53
Example 3-2 Leakage Power Calculation when State Definitions are Exclusive	55
Example 3-3 Leakage Power Calculation when State Definitions are Not Exclusive	55
Example 3-4 Leakage Power Calculation when Cell Leakage Power Information Missing	56
Example 4-1 Reading a TCF File for an Instance of a Design Loaded in RTL Compiler	67
Example 4-2 Extracting Switching Activities of an Instance from a Full Chip TCF File	69
Example 4-3 Reading a VCD File for an Instance of a Design Loaded in RTL Compiler	73
Example 4-4 Extracting Switching Activities of an Instance from a Full Chip VCD File	74
Example 5-1 Script for RTL Power Analysis	87
Example 5-2 Detailed Hierarchical RTL Power Analysis Report	87
Example 5-3 Detailed Flat RTL Power Analysis Report	88
Example 5-4 Reporting Clock Tree Power	90
Example 5-5 Reporting Clock Tree Power for Clock TCK	91
Example 5-6 Probability and Toggle Rate of Net Driven by a Clock Port	101
Example 6-1 RTL of Flip-Flops without Inferred Synchronous Reset	110
Example 6-2 RTL of Flip-Flops Inferred with a Synchronous Reset	111
Example 6-3 RTL of Flip-Flops Inferred with an Asynchronous Reset	112
Example 6-4 Example of Hierarchical Clock Gating	113
Example 6-5 Clock-Gating if Only Clock-Gating Module for Positive-Edge Triggered Registers	119
Example 6-6 Clock Gating with Clock-Gating Modules for Positive and Negative-Edge Triggered Registers	120
Example 6-7 Clock Gating Using Integrated Clock-Gating Cell	127
Example 6-8 Clock Gating Using Discrete Clock-Gating Logic	128
Example 6-9 Clock-Gating What-If	134
Example 6-10 Decloning Clock-Gating Instances	153
Example 6-11 Removing Specific Clock-Gating Instance	156

Low Power in Encounter RTL Compiler

List of Examples

<u>Example 6-12 Removing Clock-Gating Logic in a Hierarchical Design</u>	156
<u>Example 6-13 Creating Shared Clock-Gating Logic Using Common Enable</u>	160
<u>Example 6-14 Transform a Single Stage of Clock Gating into Multi-Stage Clock Gating</u> .	165
<u>Example 7-1 Nested enable Statements</u>	179
<u>Example 7-2 Combined enable Statement</u>	179
<u>Example 7-3 Inserting Operand Isolation</u>	182
<u>Example 8-1 Script for Common Leakage Optimization Flow</u>	190
<u>Example 9-1 Script for Recommended Dynamic Optimization Flow</u>	202
<u>Example 10-1 Script for Common Top-Down CPF-MSV Flow</u>	218
<u>Example 11-1 Script for Common Top-Down CPF-PSO Flow</u>	243
<u>Example 12-1 Script for Common Top-Down CPF-DVFS Flow</u>	271

Preface

- [About This Manual](#) on page 20
- [Additional References](#) on page 20
- [How to Use the Documentation Set](#) on page 21
- [Reporting Problems or Errors in Manuals](#) on page 22
- [Customer Support](#) on page 22
- [Messages](#) on page 23
- [Man Pages](#) on page 23
- [Command-Line Help](#) on page 24
- [Documentation Conventions](#) on page 26

About This Manual

This manual describes how to use the low power (RC-LP) engine in the Encounter™ RTL Compiler Ultra environment. To use this manual, you should be familiar with IC power consumption concepts and with the RTL Compiler software.

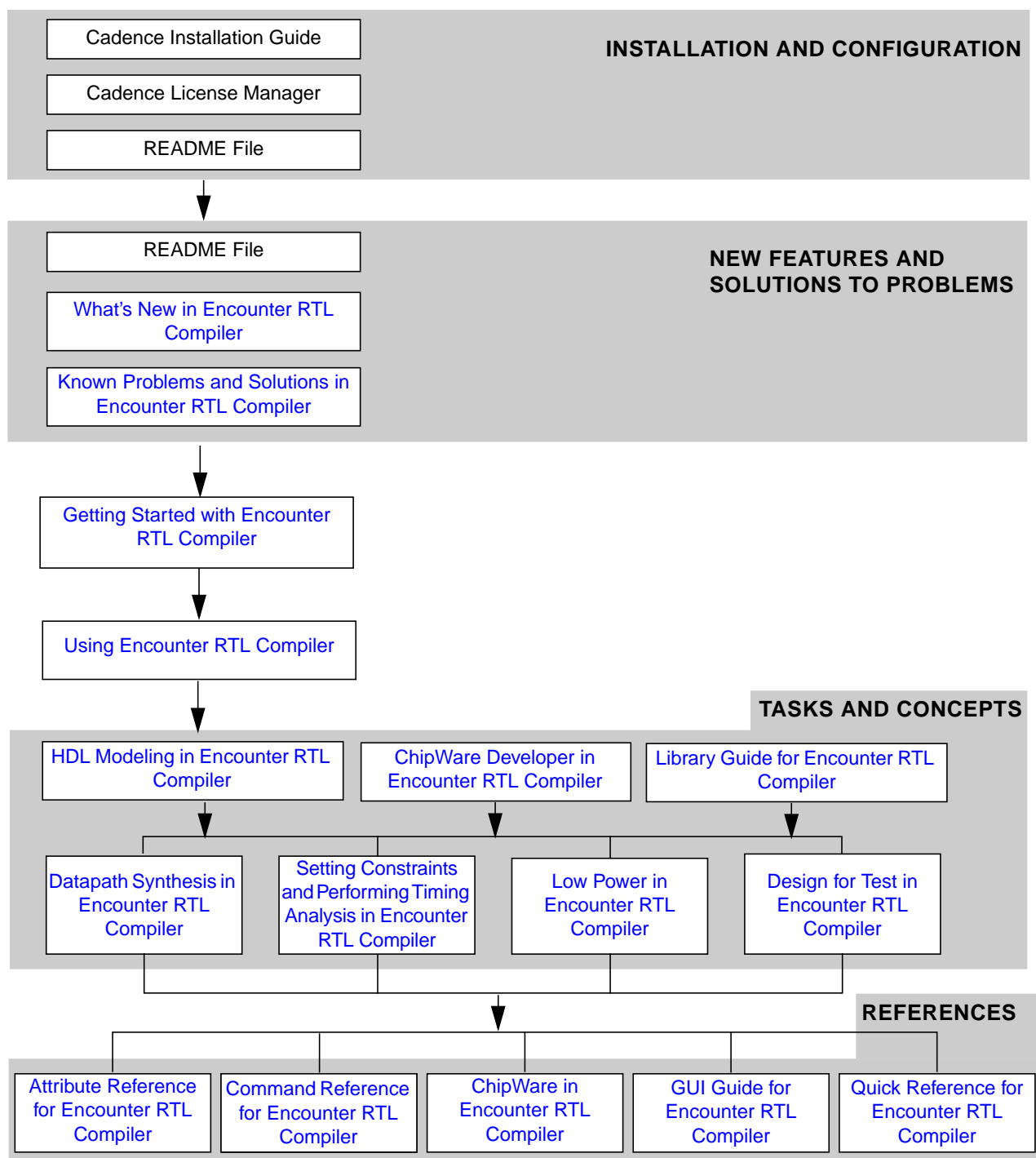
Additional References

The following sources are helpful references, but are not included with the product documentation:

- TclTutor, a computer aided instruction package for learning the Tcl language:
<http://www.msen.com/~clif/TclTutor.html>.
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1364-1995)
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1364-2001)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1987)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1993)

Note: For information on purchasing IEEE specifications go to <http://shop.ieee.org/store/> and click on *Standards*.

How to Use the Documentation Set



Reporting Problems or Errors in Manuals

The Cadence® Help online documentation, lets you view, search, and print Cadence product documentation. You can access Cadence Help by typing `cdnshelp` from your Cadence tools hierarchy.

Contact Cadence Customer Support to file a PCR if you find:

- An error in the manual
- An omission of information in a manual
- A problem using the Cadence Help documentation system

Customer Support

Cadence offers live and online support, as well as customer education and training programs.

SourceLink Online Customer Support

SourceLink® online customer support offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give you step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, service request tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

For more information on SourceLink go to:

<http://www.cadence.com/support/sourcelink.aspx>

Other Support Offerings

- **Support centers**—Provide live customer support from Cadence experts who can answer many questions related to products and platforms.
- **Software downloads**—Provide you with the latest versions of Cadence products.
- **Education services**—Offers instructor-led classes, self-paced Internet, and virtual classroom.
- **University software program support**—Provides you with the latest information to answer your technical questions.

For more information on these support offerings go to:

<http://www.cadence.com/support>

Messages

From within RTL Compiler there are two ways to get information about error messages.

- Use the `report messages` command.

For example:

```
rc:/> report messages
```

This returns the detailed information for each message output in your current RTL Compiler run. It also includes a summary of how many times each message was issued.

- Use the `man` command.

Note: You can only use the `man` command for messages within RTL Compiler.

For example, to get more information about the “TIM-11” message, type the following command:

```
rc:/> man TIM-11
```

If you do not get the details that you need or do not understand a message, either contact Cadence Customer Support to file a PCR or email the message ID you would like improved to:

`rc_msg_improvement@cadence.com`

Man Pages

In addition to the Command and Attribute References, you can also access information about the commands and attributes using the man pages in RTL Compiler. Man pages contain the same content as the Command and Attribute References.

To use our man pages from the UNIX shell:

1. Set your environment to view the correct directory:

```
setenv MANPATH $CDN_SYNTH_ROOT/share/synth/man
```

2. Enter the name of the command or attribute that you want:

```
❑ man check_dft_rules
```

```
❏ man cell_leakage_power
```

You can also use the `more` command, which behaves like its UNIX counterpart. If the output of a manpage is too small to be displayed completely on the screen, use the `more` command to break up the output. Use the spacebar to page forward, like the UNIX `more` command.

```
rc:/> more man synthesize
```

Command-Line Help

You can get quick syntax help for commands and attributes at the RTL Compiler command-line prompt. There are also enhanced search capabilities so you can more easily search for the command or attribute that you need.

Note: The command syntax representation in this document does not necessarily match the information that you get when you type `help command_name`. In many cases, the order of the arguments is different. Furthermore, the syntax in this document includes all of the dependencies, where the help information does this only to a certain degree.

If you have any suggestions for improving the command-line help, please e-mail them to:

rc_pubs@cadence.com

Getting the Syntax for a Command

Type the `help` command followed by the command name.

For example:

```
rc:/> help path_delay
```

This returns the syntax for the `path_delay` command.

Getting the Syntax for an Attribute

Type the following:

```
rc:/> get_attribute attribute_name * -help
```

For example:

```
rc:/> get_attribute max_transition * -help
```

This returns the syntax for the `max_transition` attribute.

Searching for Attributes

You can get a list of all the available attributes by typing the following command:

```
rc:/> get_attribute * * -h
```

You can type a sequence of letters after the `set_attribute` command and press `Tab` to get a list of all attributes that contain those letters.

```
rc:/> set_attr li
ambiguous "li": lib_lef_consistency_check_enable lib_search_path libcell
liberty_attributes libpin library library_domain line_number
```

Searching For Commands When You Are Unsure of the Name

You can use help to find a command if you only know part of its name, even as little as one letter.

- If you only know the first few letters of a command you can get a list of commands that begin with that letter.

For example, to get a list of commands that begin with “ed”, you would type the following command:

```
rc:/> ed* -h
```

- You can type a single letter and press `Tab` to get a list of all commands that contains that letter.

For example:

```
rc:/> c <Tab>
```

This returns the following commands:

```
ambiguous "c": cache_vname calling_proc case catch cd cdsdoc change_names
check_dft_rules chipware clear clock clock_gating clock_ports close cmdExpand
command_is_complete concat configure_pad_dft connect_scan_chains continue
cwd_install ..
```

- You can also type a sequence of letters and press `Tab` to get a list of all commands that contain those letters.

For example:

```
rc:/> path_<Tab>
```

This returns the following commands:

```
ambiguous command name "path_": path_adjust path_delay path_disable path_group
```

Documentation Conventions

The list below describes the syntax conventions used for the RTL Compiler commands and attributes.

<code>literal</code>	Nonitalic words indicate keywords that you must enter literally. These keywords represent command or option names.
<code>arguments and options</code>	Words in italics indicate user-defined arguments or information for which you must substitute a name or a value.
<code> </code>	Vertical bars (OR-bars) separate possible choices for a single argument.
<code>[]</code>	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one.
<code>{ }</code>	Braces indicate that a choice is required from the list of arguments separated by OR-bars. Choose one from the list. <code>{ argument1 argument2 argument3 }</code> Braces, used in Tcl commands, indicate that the braces must be typed in.
<code>{ }</code>	Braces, used in Tcl commands, indicate that the braces must be typed in.
<code>...</code>	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, <code>[argument]...</code>), you can specify zero or more arguments. If the three dots are used without brackets (<code>argument...</code>), you must specify at least one argument, but can specify more.
<code>#</code>	The pound sign precedes comments in command files.

Introduction

This chapter discusses the following topics:

- Today's Power Challenges on page 28
- Traditional Low Power Synthesis Features on page 28
 - Power Analysis on page 29
 - Power Optimization on page 29
- Support for Advanced Power Management Techniques through the Common Power Format on page 33
- Required Files for Low Power Synthesis on page 34

Today's Power Challenges

With the remarkable success and growth of personal computing devices, consumer electronics, and wireless communication systems, there is an urgent demand for high-speed computation and complex functionality. However, power dissipation has also become a critical design concern. Because of shrinking chip size, the density per unit area is increasing, and so is the cost of packaging and cooling strategies of high-performance processors, such as those processors with 500-3000 MHz clocks.

These issues shift the focus from design performance and area concerns to a need for lower power and consideration for power dissipation. With today's technology shrinking down to 90 nm and below, the importance of leakage power in power dissipation is also increasing.

The challenge is to meet these needs without compromising overall chip performance. Power consumption in design modules must be identified and their power consumption reduced. These are the goals for the low power (RC-LP) engine in the Encounter™ RTL Compiler Ultra environment.

Traditional Low Power Synthesis Features

When your Register-Transfer Level (RTL) netlist is ready for synthesis, low power synthesis (LP) is performed as part of the normal synthesis process. The following sections introduce the techniques that the low power (RC-LP) engine offers for

- [Power Analysis](#) on page 29
- [Power Optimization](#) on page 29

Power Analysis

Chapter 3, “Power Analysis Concepts” explains how the different power components are calculated.

All power components vary with the switching activity in the circuit. Chapter 4, “Providing Switching Activity Information” explains the different ways you can provide that information.

To analyze the power you can

- Generate a detailed power report, including detailed power reports on a hierarchical or flat instance
- Query the power of instances and nets

You can analyze power at different stages in the flow. By performing RTL power analysis you can verify the power budget and guide the power optimization.

For more information on power analysis, refer to Chapter 5, “Power Analysis.”

Power Optimization

The main features for power optimization are:

- Clock Gating
- Operand Isolation
- Leakage Power Optimization
- Dynamic Power Optimization

You can selectively enable each of these features.

Clock Gating

The low power (RC-LP) engine explores the design to determine where it can make potential power savings by inserting clock-gating logic for register banks.

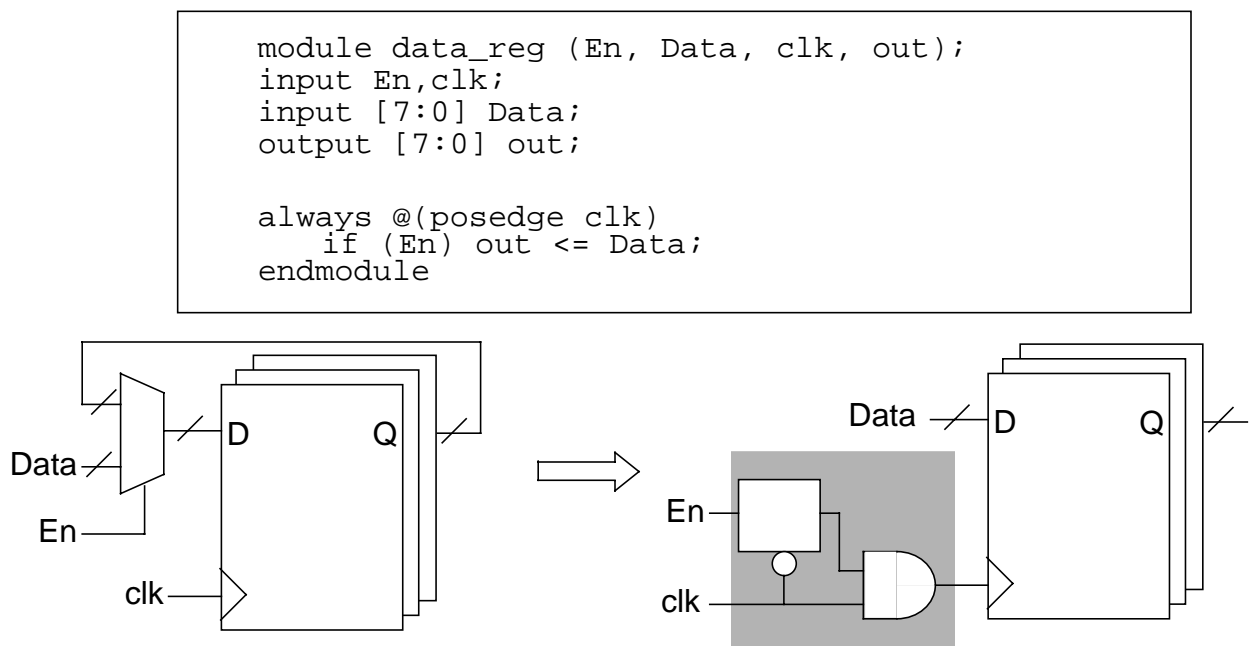
In many designs, data is loaded into registers infrequently, but the clock signal continues to switch at every clock cycle often driving a large capacitive load. You can save a significant amount of power by identifying when the registers are inactive and by disabling the clock during these periods.

As illustrated in Figure 1-1, to reduce unnecessary clock toggles, the RC-LP engine inserts clock-gating logic for a group of registers that are enabled by the same synchronous signal.

The RC-LP engine can insert clock gating when starting either from RTL or from a mapped netlist. Some other advanced clock-gating features include hierarchical clock-gating insertion and clock-gating decloning.

The RC-LP engine also supports test synthesis by adding the controllability and observability logic to the clock-gating logic. For more information, refer to [Chapter 6, “Clock Gating.”](#)

Figure 1-1 Clock Gating Concept



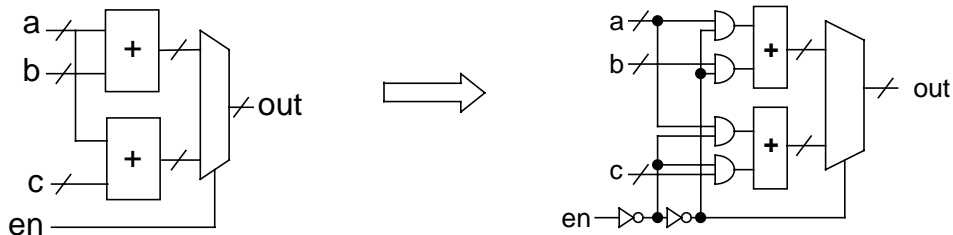
Operand Isolation

Where clock-gating is effective to save power on sequential elements, it cannot prevent power loss in functional blocks. Significant power is lost when computations continue even if the results are not used.

As illustrated in [Figure 1-2](#) on page 31, the RC-LP engine can insert operand-isolation logic for datapath elements or (hierarchical) combinational modules whose output is controlled by the same enable signal.

Figure 1-2 Operand Isolation Concept

```
module test (en, a, b, c, out);  
  input en;  
  input [7:0] a, b, c;  
  output [8:0] out;  
  
  assign out = en? a+b : a+c;  
endmodule
```



For more information, refer to [Chapter 7, “Operand Isolation.”](#)

Leakage Power Optimization

Traditionally, static leakage power dissipation is only a small part of the overall power consumption. However, with the advances in process technology, static leakage power becomes more and more significant. As a result, accurate leakage power estimation and efficient leakage power optimization techniques become crucial for any low power solutions.

The availability of two or more threshold voltages on the same chip (multiple-V_{th} process) provides a new opportunity to make trade-offs between power and performance.

For more information, refer to [Chapter 8, “Leakage Power Optimization.”](#)

Dynamic Power Optimization

Dynamic power is the power dissipated by an instantaneous short-circuit connection between the voltage supply and the ground when the gate transitions, and the switching power dissipated when charging or discharging internal and net capacitances.

During global optimization in RTL Compiler, timing and power optimization can be performed simultaneously throughout the synthesis flow, including during global mapping, remapping, and incremental optimization.

For more information, refer to [Chapter 9, “Dynamic Power Optimization.”](#)

Support for Advanced Power Management Techniques through the Common Power Format

The Common Power Format (CPF) addresses the current limitation in the design automation tool flow by enabling the capture of the designer's intent for advanced power management techniques. CPF captures all design and technology-related power constraints in a single file format for use throughout the RTL to GDSII design flow including verification, validation, synthesis, test, physical implementation, and signoff analysis.

Advanced power management techniques include

- Using multiple supply voltages in a design

This is one of the most effective approaches to reduce the dynamic power dissipation of a design. The dynamic power dissipation is a quadratic function of the supply voltage. Using a high supply voltage for blocks on timing-critical paths, while using a low supply voltage for blocks along the non-critical timing paths can result in a significant dynamic power saving while still meeting the timing requirements.

For more information about the recommended flow for MSV designs, refer to [Chapter 10, “Using CPF for Multiple Supply Voltage Designs.”](#)

- Using the Power Shut Off (PSO) methodology.

Using this methodology, some portions of the design are switched on and off as needed to save leakage and dynamic power.

For more information about the recommended flow for designs using the PSO methodology, refer to [Chapter 11, “Using CPF for Designs Using Power Shutoff Methodology.”](#)

Note: The low power (RC-LP) engine can perform power domain-aware power analysis.

- Using dynamic voltage frequency scaling (DVFS)

Dynamic voltage frequency scaling (DVFS) reduces the power in the chip by scaling down the voltage and frequency when peak performance is not required.

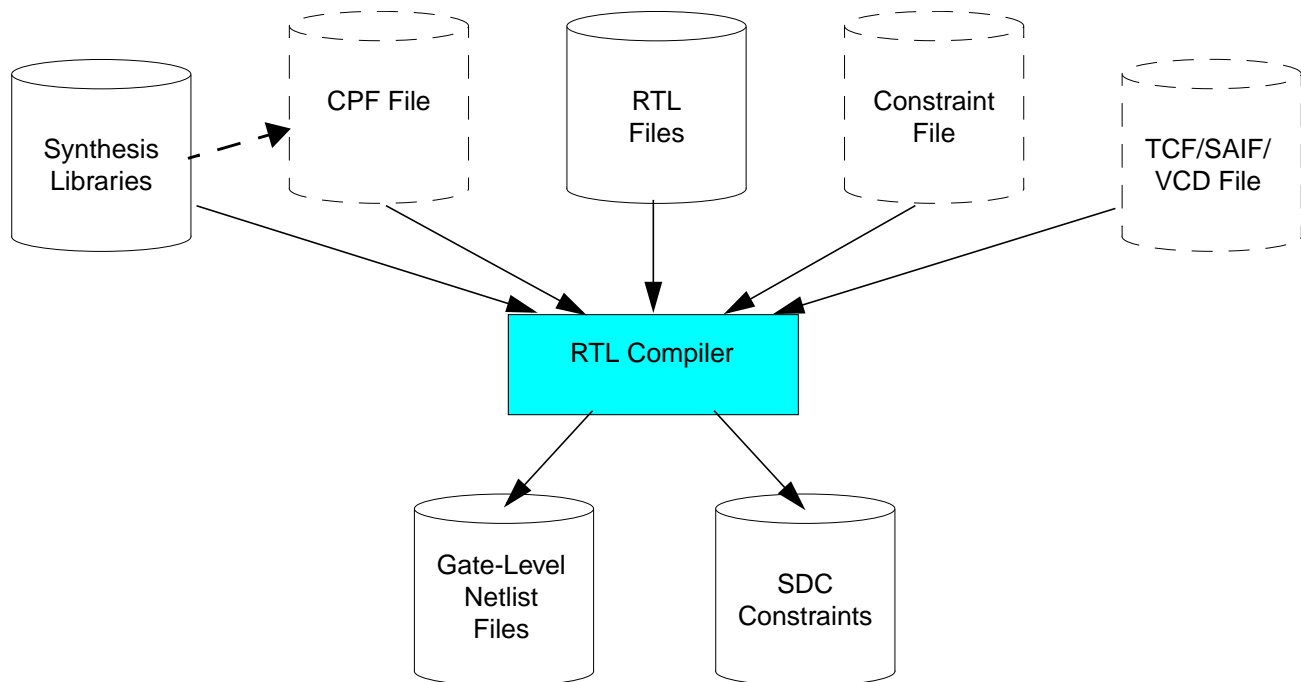
A design using DVFS can be seen as a special case of an MSV design operating in multiple design modes.

- ❑ In a pure MSV design different portions of the design operate on different voltages and these portions *remain* operating at their respective operating voltage.
- ❑ In a DVFS design, in addition some portions can dynamically *change* to other voltages depending on the design mode or can even be switched off.

Required Files for Low Power Synthesis

Figure 1-3 shows the data flow for RC-LP.

Figure 1-3 Data Flow for Low Power Synthesis in RTL Compiler Environment



You need the following files when you start the RTL Compiler flow:

- A synthesis library in Liberty format containing
 - ❑ Timing information
 - ❑ Internal cell power dissipation information
 - ❑ Leakage power dissipation information
 - ❑ (Optional) Integrated clock-gating cells

For more information about the Liberty low power requirements, refer to [Low Power Requirements](#) in the *Library Guide for Encounter RTL Compiler*.

- RTL files

The Verilog files can contain structural code for combining lower level modules, behavioral design specifications, or RTL implementations.

- CPF file if you like to use advanced power management techniques

Low Power in Encounter RTL Compiler

Introduction

- Constraints file

You can enter constraints manually in the RTL Compiler shell, include a constraints file, or read in SDC Tcl constraints.

- TCF or SAIF file

You can read in net switching activities from a TCF or SAIF file for more accurate power analysis.

Low Power in Encounter RTL Compiler

Introduction

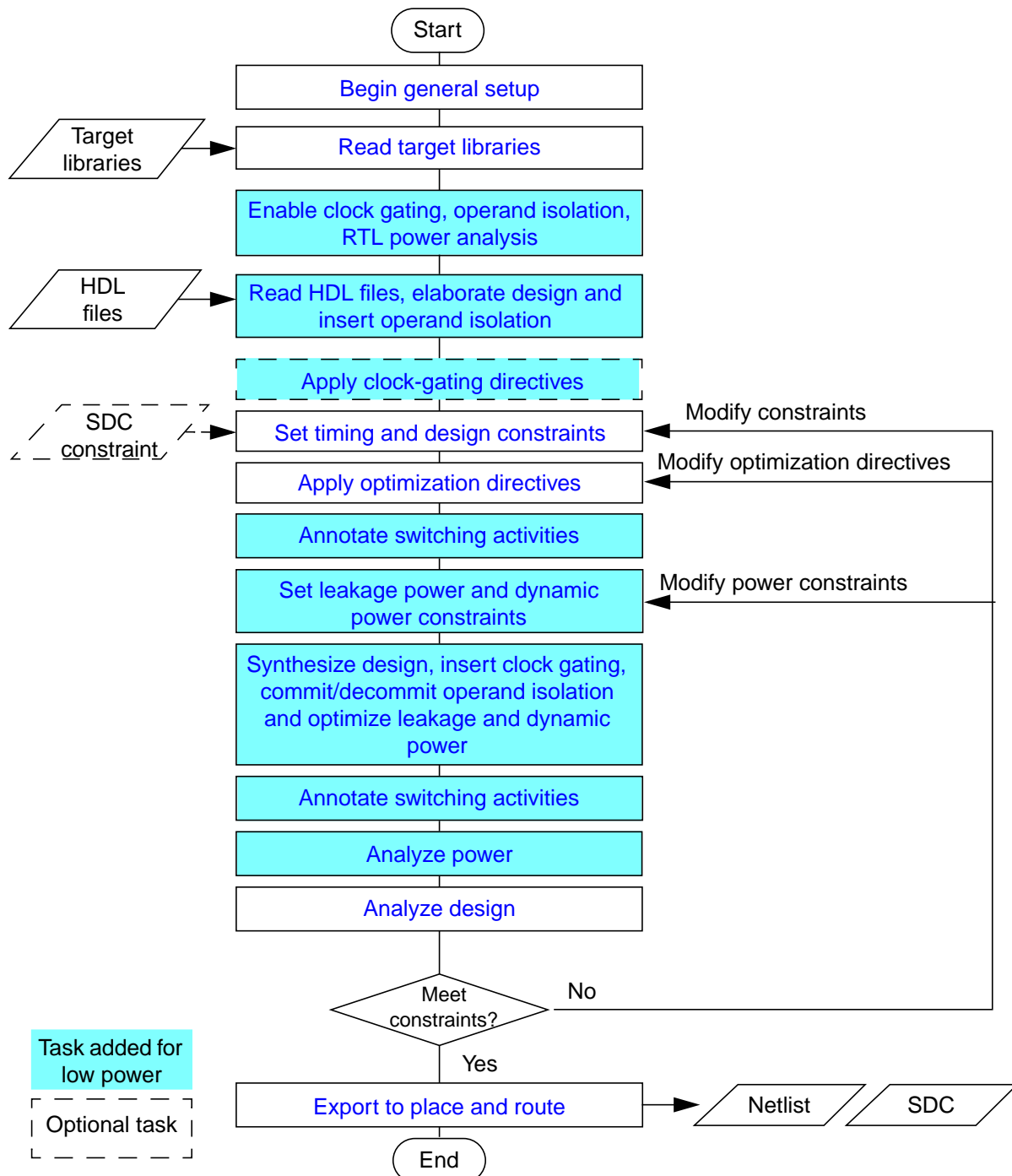
Recommended Flow

- [Synthesis Flow with Low Power Features](#) on page 38
- [Flow Steps](#) on page 39
 - ❑ [Begin Setup](#) on page 39
 - ❑ [Read Target Libraries](#) on page 40
 - ❑ [Enable Clock Gating, Operand Isolation, RTL Power Analysis](#) on page 40
 - ❑ [Read HDL Files, Elaborate Design, and Insert Operand Isolation](#) on page 40
 - ❑ [Apply Clock-Gating Directives](#) on page 41
 - ❑ [Set Timing and Design Constraints](#) on page 41
 - ❑ [Apply Optimization Directives](#) on page 42
 - ❑ [Set Leakage Power and Dynamic Power Constraints](#) on page 42
 - ❑ [Annotate Switching Activities](#) on page 43
 - ❑ [Synthesize Design](#) on page 43
 - ❑ [Annotate Switching Activities](#) on page 44
 - ❑ [Analyze Power](#) on page 44
 - ❑ [Analyze Design](#) on page 44
 - ❑ [Export to Place and Route](#) on page 45

Synthesis Flow with Low Power Features

Steps are explained in [Flow Steps](#) on page 39.

Figure 2-1 Top-Down Low Power Synthesis Flow



Flow Steps

This section briefly describes the steps in the top-down synthesis flow, which includes clock-gating insertion, leakage power optimization and power analysis.

Chapter 10, “Using CPF for Multiple Supply Voltage Designs,” covers the low power flow when the design uses multiple supply voltages without considering shutting down some portions of the design.

Chapter 11, “Using CPF for Designs Using Power Shutoff Methodology,” describes the recommended flow for a design using the power shut off methodology.

Chapter 12, “Using CPF for Designs Using Dynamic Voltage Frequency Scaling,” describes the recommended flow for a design using the dynamic voltage frequency scaling methodology.

Chapter 13, “Other Flows” covers the following flows:

- Recommended Bottom-Up Clock Gating Flow with DFT
- Scan Insertion after Clock-Gating Insertion
- Power Optimization when Starting with Mapped Netlist
- State-Retention Cell Replacement when Starting with Mapped Netlist

Begin Setup

The default search path for libraries, scripts, and HDL files is the directory in which RTL Compiler is invoked.

- To specify the paths for RTL Compiler to search for libraries, scripts, and HDL files, set the following attributes:

```
set_attribute lib_search_path ...  
set_attribute hdl_search_path ...  
...
```

For more information on the setup, see Using the Technology Library in *Using Encounter RTL Compiler*.

Read Target Libraries

After you set the library search path, specify the target technology library for synthesis by setting the following attribute:

```
set_attribute library library_list /
```

Make sure that the specified libraries contain both timing and power information.

Note: Loading multiple libraries containing cells with different threshold voltages allows the RC-LP engine to perform timing and leakage power optimization simultaneously during synthesis.

Enable Clock Gating, Operand Isolation, RTL Power Analysis

1. To ensure that clock-gating logic is inserted during synthesis, set the following attributes:

```
set_attribute lp_insert_clock_gating true /  
set_attr lp_clock_gating_prefix string / ;#optional
```

For more information on clock-gating insertion, refer to [Chapter 6, “Clock Gating.”](#)

2. To ensure that operand-isolation logic is inserted during synthesis, set the following attributes:

```
set_attribute lp_insert_operand_isolation true /  
set_attr lp_operand_isolation_prefix string / ;#optional
```

For more information on operand isolation, refer to [Chapter 7, “Operand Isolation.”](#)

3. To enable RTL power analysis, set the following attribute:

```
set_attr hdl_track_filename_row_col true /  
set_attribute lp_power_unit mW / ;#optional
```

For more information on RTL power analysis, refer to [Chapter 5, “Power Analysis.”](#)

Read HDL Files, Elaborate Design, and Insert Operand Isolation

1. After you read in the libraries, read the HDL files into RTL Compiler:

```
read_hdl verilog_files
```

When you issue a `read_hdl` command, RTL Compiler reads the files and performs syntax checks. RTL Compiler can read in Verilog and VHDL files.

For more information on reading HDL files, see [Loading Files](#) in *Using Encounter RTL Compiler*.

2. To transform the raw HDL text into a suitable form for synthesis, elaborate the design:

```
elaborate
```


The `elaborate` command automatically elaborates the top-level design and all of its references. During elaboration, the RC-LP engine identifies

- ❑ The registers for clock gating and determines when those registers are inactive
- ❑ The datapath block candidates (such as adders and multipliers) for operand isolation and inserts operand isolation instances

For more information on elaboration, refer to [Elaborating the Design](#) in *Using Encounter RTL Compiler*.

Apply Clock-Gating Directives

To select a clock-gating integrated cells that is available in the library, use the following design attributes:

```
lp_clock_gating_cell  
lp_clock_gating_add_obs_port  
lp_clock_gating_add_reset  
lp_clock_gating_control_point  
lp_clock_gating_style
```

Note: If your library does not contain the integrated clock-gating cell that you request through the attributes, the RC-LP engine will use regular library cells to instantiate the clock-gating module that corresponds to the clock-gating directives.

For more information, refer to [Controlling Selection of Clock-Gating Logic](#) on page 115.

To control the clock gating, use the following design attributes:

```
lp_clock_gating_max_flops  
lp_clock_gating_min_flops  
lp_clock_gating_exclude
```

For more information, refer to [Controlling Insertion of Clock-Gating Logic](#) on page 130.

Note: All attributes have default settings. If you are fine with these settings, you do not need to set any of these attributes.

Set Timing and Design Constraints

After reading and elaborating your design, you must provide timing and design constraints, such as operating conditions, clock waveforms, and I/O timing.

You can apply constraints in several ways:

- Enter them manually in the RTL Compiler shell.
- Include a constraints file.

Low Power in Encounter RTL Compiler

Recommended Flow

- Read in SDC Tcl constraints.

For more information on setting design constraints, see [Applying Constraints](#) in the *Using Encounter RTL Compiler*.

Apply Optimization Directives

In addition to applying design constraints, you may need to use further optimization strategies to get the desired performance goals from synthesis. You can

- Preserve instances and modules
- Group and ungroup instances or subdesigns
- Control boundary optimization of hierarchical instances
- Map to complex sequential cells
- Flatten arrays and memories
- Specify ideal nets
- Disable timing arcs
- Create path groups and cost groups
- Optimize total negative slack
- Make DRC the highest priority

For more information on optimization strategies and related commands, see [Defining Optimization Settings](#) in the *Using Encounter RTL Compiler*.

Set Leakage Power and Dynamic Power Constraints

To enable leakage power optimization, set the following attribute:

```
set_attribute max_leakage_power float /designs/design
```

To enable dynamic power optimization, set the following attribute:

```
set_attribute max_dynamic_power float /designs/design
```

The RC-LP engine continues power optimization until the power of the design is smaller than the specified constraint. If both constraints are specified, leakage power optimization is optimized first by default.

To control the power optimization, use one of the following design attributes:

Low Power in Encounter RTL Compiler

Recommended Flow

```
set_attribute lp_optimize_dynamic_power_first true /designs/design
set_attribute lp_power_optimization_weight float /designs/design
```

To further minimize the leakage power you can use state-retention power gating by setting the following attributes:

```
set_attribute lp_map_to_srpq_cells true /designs/design
set_attribute lp_srpq_pg_driver string /designs/design
```

For more information on leakage power optimization, refer to [Chapter 8, “Leakage Power Optimization.”](#) For more information on dynamic power optimization, refer to [Chapter 9, “Dynamic Power Optimization.”](#)

Annotate Switching Activities

For a more accurate power optimization, annotate switching activities *before* synthesis. [Sources of Switching Activity Information](#) shows different ways you can provide the switching activities to the RC-LP engine.

Note: If not all nets were asserted, the RC-LP engine can automatically propagate the switching activities. For more information, see [Propagating Net Switching Activities](#).

To get the most accurate power analysis results, you can generate the switching activity information by simulating the design with a testbench that represents the real chip behavior. [Running Simulation to Generate TCF Files](#) explains more about the steps involved in the simulation. In this case, the switching activities are stored in a Cadence® Toggle Count Format (TCF) file. To read in this TCF file in RTL Compiler, use the following command:

```
read_tcf file
```

Synthesize Design

- After the constraints and optimization directives are set for your design, synthesize your design using the following command:

```
synthesize -to_mapped
```

Clock-gating insertion, commitment and decommitment of the operand isolation instances, and leakage and dynamic power optimization occur automatically during mapping and optimization.

- During mapping, the low power (RC-LP) engine inserts one gating logic per register bank with the same enable signal for clocks in the inactive periods. The structure of the clock-gating logic is user-defined.
- Operand isolation instances are committed based on timing and power considerations.

Low Power in Encounter RTL Compiler

Recommended Flow

- When both power constraints are set, the RC-LP engine performs timing optimization, and leakage and dynamic power optimization simultaneously during each optimization step.

Annotate Switching Activities

For a more accurate power calculation, annotate switching activities before power analysis. [Sources of Switching Activity Information](#) shows different ways you can provide the switching activities to the RC-LP engine.

If not all nets were asserted, the RC-LP engine can automatically propagate the switching activities. For more information, see [Propagating Net Switching Activities](#).

To get the most accurate power analysis results, you can generate the switching activity information by simulating the design with a testbench that represents the real chip behavior. [Running Simulation to Generate TCF Files](#) explains more about the steps involved in the simulation. In this case, the switching activities are stored in a Cadence® Toggle Count Format (TCF) file. To read in this TCF file in RTL Compiler, use the following command:

```
read_tcf file
```

Analyze Power

After you have specified the switching activities, you can analyze the power using the following command:

```
report power
```

For more information, refer to [Chapter 5, “Power Analysis.”](#)

Analyze Design

After optimizing the design, you can generate detailed timing and area reports using the `report` commands.

For more information on generating reports for analysis, see [Generating Reports](#) in *Using Encounter RTL Compiler* and [“Analysis Commands”](#) in the *Command Reference for Encounter RTL Compiler*.

To generate a clock-gating report use the following command:

```
report clock_gating
```

For more information, refer to [Reporting Clock-Gating Information](#) on page 146.

To generate an operand-isolation report use the following command:

Low Power in Encounter RTL Compiler

Recommended Flow

```
report operand_isolation
```

For more information, refer to [Reporting Operand Isolation Information](#) on page 182.

Export to Place and Route

As the last step in the flow, write out the gate-level netlist and the SDC file for processing in your place and route tool using the following commands:

```
write -mapped > netlist_file  
write_script > script_file  
write_sdc > sdc_file
```

For more information, refer to [Interfacing to Place and Route](#) in the *Using Encounter RTL Compiler*.

Low Power in Encounter RTL Compiler

Recommended Flow

Power Analysis Concepts

- [Power Dissipation Components](#) on page 48
- [How the RC-LP Engine Models Power Dissipation](#) on page 50
- [Power Calculation](#) on page 51

Power Dissipation Components

In the digital IC world, power dissipation for a circuit consists of:

- [Leakage Power Dissipation](#) on page 48
- [Net Power Dissipation](#) on page 48
- [Internal Power Dissipation](#) on page 49

Leakage Power Dissipation

Advances in CMOS technology have resulted in a reduction of the transistor's threshold voltage, which prevents the transistor from turning off completely. As a result, static power is dissipated, which is caused by the leakage current between source and drain.

Another component of static power dissipation is caused by the current leaks between the diffusion layers and the substrate. The power dissipation caused by these current leaks is usually much smaller than the power dissipation caused by subthreshold leakage and can therefore be ignored.

With today's technology shrinking down to 90 nm and below, the leakage power becomes more and more dominant.

Since the leakage power depends on the technology, it is entirely determined by the cell leakage power specified in the technology library. For more information, refer to [Leakage Power Specification](#) in the *Library Guide for Encounter RTL Compiler*.

Note: The RC-LP engine also supports state-dependent leakage power which is specified in the cell description section of your synthesis library.

Net Power Dissipation

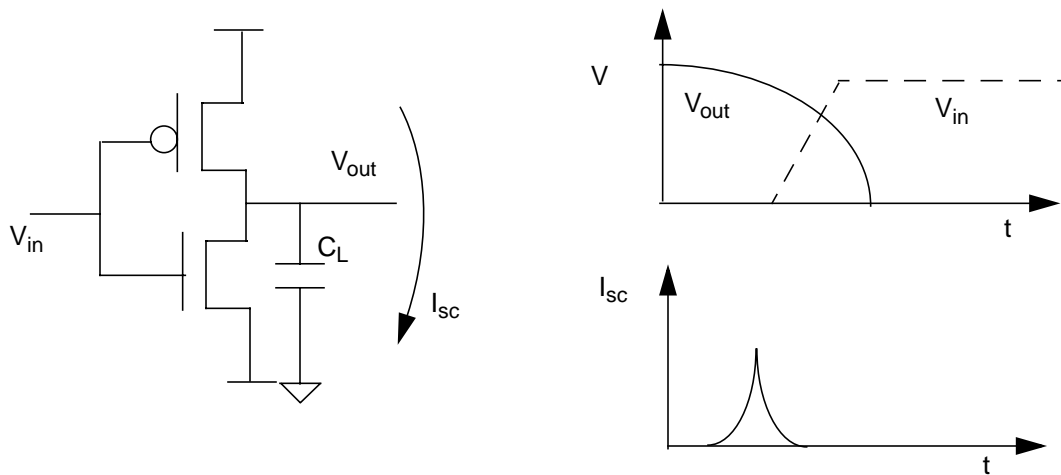
The net power (or capacitive load power) dissipation is the power consumed when charging or discharging the capacitive load which includes the net capacitance and the capacitance of the input pins.

Power dissipated by the internal capacitance is modeled as part of the internal cell power.

Internal Power Dissipation

The internal power dissipation (see Figure 3-1) is the power consumed by an instantaneous short-circuit connection between the voltage supply and the ground when the gate transitions and the internal net power dissipated when charging or discharging internal capacitances.

Figure 3-1 Short Circuit Power Dissipation

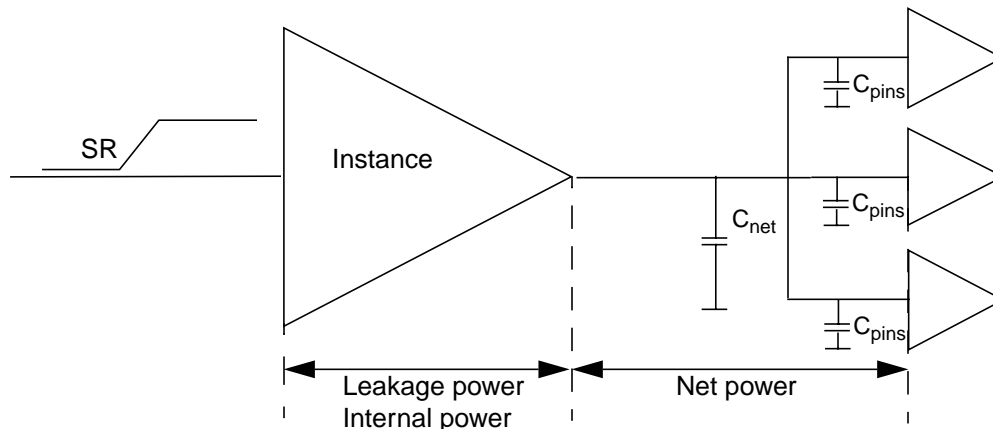


The input slew, the drive strength of the transistors, and the load driven determine the magnitude and duration of the short-circuit.

How the RC-LP Engine Models Power Dissipation

Figure 3-2 shows how the low power (RC-LP) engine models the power dissipation.

Figure 3-2 Power Dissipation Model



The power dissipated in the instance has the following components:

- *Internal power dissipation* of the cell which includes short-circuit power and the switching power dissipated when charging or discharging internal net capacitances.

The data are obtained from the power look-up tables in the library. These tables are a function of the input slew rate (SR) only, or of the input slew rate and the output load capacitance (C_L). Some libraries also have three-dimensional tables which are a function of the input slew, output load capacitance and a second output load capacitance.

- *Leakage power dissipation* of the cell.

The data is obtained from the leakage power annotation in the synthesis library. Some technology library vendors provide cells with different threshold voltages. In this case, the RC-LP engine can further optimize leakage power by utilizing high V_{th} cells (low leakage power, slow timing performance) along the non-critical timing paths, and low V_{th} cells (faster cells, higher leakage power) on timing-critical paths.

The net power includes switching power consumed by

- Charging and discharging the net capacitance
- Charging and discharging the capacitances of the pins driven by the net

Note: If the output pin capacitance is set to 0 in the library, the power consumed by this output capacitance is included in the internal power of the instance. If the output pin capacitance is set to a non-zero value, the power consumed by this capacitance will be included in the net power.

Power Calculation

The static power calculation returns the average power consumption as opposed to cycle-by-cycle power consumption. The static power calculation uses the probability and toggle rate data to estimate the power.

The following equation shows how the RC-LP engine calculates the average power dissipation for the design:

$$P_{total} = \sum_{1}^n P_{instance} + \sum_{1}^m P_{net}$$

$$\sum_{1}^n P_{instance} = \sum_{1}^n P_{leakage} + \sum_{1}^n P_{internal}$$

where

- $P_{leakage}$ is the leakage power in a cell, which is obtained from the technology library. The RC-LP engine supports both constant and state-dependent leakage power models.

The unit is determined by the `leakage_power_scale_in_nW` attribute, which is derived from the `leakage_power_unit` of the technology library.

- $P_{internal}$ is the internal power of a cell, which is obtained from the technology library. The RC-LP engine supports all internal power models in Liberty, including state-dependent and path-dependent internal power models.

The units used for the internal cell power are derived from other library units as follows:

$$\text{power_unit} = \text{cap_unit} \times \frac{\text{volt_unit}^2}{\text{time_unit}}$$

- P_{net} is the net power.
- n is the number of cells in the design.
- m is the number of nets in the design.

For more information, refer to [Units for Internal Cell Power](#) in the *Library Guide for Encounter RTL Compiler*

Internal Cell Power Calculation

The internal cell power is usually modeled in the library as a look-up table in function of the input slew rate (SR) and the output load capacitance (CL). Some libraries also have three-dimensional power tables which are function of the input slew, output load capacitance and a second output load capacitance.

Instead of one table, a cell can contain two tables: one table for a rise transition and one table for a fall transition.

To model internal cell power more accurately as technology advances to 90nm and below, internal cell power can even be defined per state and per path.

Taking into account path dependencies, the internal cell power is determined as follows:

$$P_{internal} = \sum_{\text{per arc}} (TR_{arc_{ij}} \times \Phi(S_i, C_j)) + \sum_{\text{per pin}} (TR_i \times \Phi(S_i))$$

where

- TR is the effective toggle rate of an arc or pin. The effective toggle rate of arc_{ij} depends on the probability that the arc gets activated and on the toggle rate on input pin i . The probability that the arc gets activated is determined by the function of output pin j and the probabilities of the other input pins. Refer to [Chapter 4, “Providing Switching Activity Information,”](#) for more information on how the toggle rate information can be obtained.
- Φ is the power which is obtained from a look-up table in the technology library. For more information on how internal power is modelled in the library, refer to [Internal Power Specification](#) in the *Library Guide for Encounter RTL Compiler*.
- S_i is the slew of input i causing a toggle on output j .
- C_j is the load capacitance of output j .

[Example 3-1](#) on page 53 shows how the power of one arc in a two-input AND gate is calculated.

Example 3-1 Internal Power Calculation

The following partial library description for cell AND2D1 shows the internal power information for arc from pin A1 (see `related_pin` statement) to pin Z (see `pin(Z)` statement).

```
cell(AND2D1) {
  ...
  pin(Z) {
    ...
    function : "( A1 & A2 )" ;
    direction : output ;
    ...
    internal_power() {
      rise_power(li3X4) {
        index_1("0.012000, 0.024000, 0.048000") ;
        index_2("0.042, 0.168, 0.336, 1.100") ;
        values("0.0061, 0.0059, 0.0061, 0.0086",\
              "0.0061, 0.0059, 0.0061, 0.0086",\
              "0.0062, 0.0060, 0.0062, 0.0085";
      }
      fall_power(li3X4) {
        index_1("0.012000, 0.024000, 0.048000") ;
        index_2("0.042, 0.168, 0.336, 1.100") ;
        values("0.0057, 0.0056, 0.0059, 0.0085",\
              "0.0058, 0.0057, 0.0059, 0.0084",\
              "0.0058, 0.0057, 0.0060, 0.0084";
      }
      when : "A2" ;
    }
    related_pin : A1 ;
  }
  ...
  internal_power() {
    ...
    related_pin : A2 ;
  }
}
...
```

Assume the slew at input A1 is 18 ps and net capacitance seen at pin Z is 0.336 pF. The power of the arc from pin A1 to pin Z is calculated based on the toggle rate of pin A1, the probability that the arc is activated, and the internal power (Φ) in function of the input slew and load.

$$\text{toggle_rate}_{A1} \times \text{probability}_{A1 \rightarrow Z} \times \Phi_{A1 \rightarrow Z}(0.0018, 0.336)$$

Because the input slew does not correspond to an index value, the value of Φ must be interpolated between the values shown in bold. If the `internal_power` group in the library has a `fall_group` and a `rise_group`, the power is determined as the average of the fall and rise power. In this case the value of Φ is calculated as:

$$\begin{aligned} \Phi &= 0.5 \times \text{rise_power}(0.0018, 0.336) + 0.5 \times \text{fall_power}(0.0018, 0.336) \\ &= 0.5 \times 0.0061 + 0.5 \times 0.0059 \end{aligned}$$

Net Power Calculation

The net power (caused by charging the net capacitance) is determined as follows:

$$P_{net} = \frac{1}{2} \times C_L \times V^2 \times TR$$

where

- C_L is the capacitance in femtofarads (default). The capacitance is the sum of the capacitances of the net and of the input pins driven by the net. This information is stored in the pin capacitance and wire capacitance attributes.
- V is the supply voltage in volts. This information is specified in the voltage attribute
- TR is the toggle rate. The unit is determined by the lp_toggle_rate_unit attribute. Refer to Chapter 4, “Providing Switching Activity Information,” for more information on how the RC-LP engine obtains the toggle rate information.

Leakage Power Calculation

The leakage power is usually modelled in the library as a constant. However, as technology advances to 90nm and below, some libraries specify cell leakage power as a function of the input state to model leakage power more accurately.

For more information on how leakage power can be modelled in the library and how the `.lib` attributes affect the leakage power calculation, refer to Leakage Power Specification in the *Library Guide for Encounter RTL Compiler*.

The leakage power for a cell is determined as follows:

$$P_{cell_leakage} = \sum_{state=1}^k (P_{state_leakage} \times probability_{state})$$

where

- k is the number of states for a cell.
- $P_{state_leakage}$ is the leakage of the cell in that state.
- $probability_{state}$ is the probability that the cell is in this state.

The following examples show how the cell leakage power is calculated for a two-input AND gate.

Example 3-2 Leakage Power Calculation when State Definitions are Exclusive

The AND2X2 cell can have four possible states. In this example the leakage power is specified for three states through the `leakage_power` statements. The cell also has a `cell_leakage_power` statement. The value specified in this statement is used for the missing fourth state when A&B are applied.

```
cell (AND2X2) {  
...  
    cell_leakage_power : YYYY;  
    leakage_power() {  
        when : "!A & !B";  
        value : yy11;  
    }  
    leakage_power() {  
        when : "!A & B";  
        value : yy22;  
    }  
    leakage_power() {  
        when : "A & !B";  
        value : yy33;  
    }  
...  
}
```

Assuming that the probability for A to be 1 is 0.6 and the probability for B to be 1 is 0.3, the leakage power can be calculated as follows:

$$\text{prob}(!A) \times \text{prob}(!B) \times yy11 + \text{prob}(!A) \times \text{prob}(B) \times yy22 + \text{prob}(A) \times \text{prob}(!B) \times yy33 + \text{prob}(A) \times \text{prob}(B) \times YYYY$$

or

$$0.4 \times 0.7 \times yy11 + 0.4 \times 0.3 \times yy22 + 0.6 \times 0.7 \times yy33 + 0.6 \times 0.3 \times YYYY$$

Example 3-3 Leakage Power Calculation when State Definitions are Not Exclusive

In this example the leakage power is again specified for three conditions through the `leakage_power` statements. In this case, the conditions are not mutually exclusive.

```
cell (AND2x2 ) {  
...  
    area : 4.0000;  
    cell_leakage_power : 0.1610;  
    leakage_power () {  
        when : "!A ^ !B" ;  
        value : 0.1620;  
    }  
    leakage_power () {  
        when : "!A" ;  
        value : 0.1630;  
    }  
    leakage_power () {  
        when : "!B" ;  
        value : 0.1640;  
    }  
...  
}
```

Low Power in Encounter RTL Compiler

Power Analysis Concepts

If the RC-LP engine detects non-mutually exclusive `when` conditions, it prints out a warning message and ignores the state-dependent leakage power and uses the default cell leakage power.

In this case, the RC-LP engine will return 0.1610 for the cell leakage power.

Example 3-4 Leakage Power Calculation when Cell Leakage Power Information Missing

Assume the library has the following power-related specifications.

```
library(my_lib) {  
    ...  
    /* unit attributes */  
    time_unit : "1ns";  
    voltage_unit : "1V";  
    capacitive_load_unit(1,pf);  
    ...  
    leakage_power_unit : "1nW";  
    /* default attributes */  
    default_leakage_power_density : 0.5;;  
}
```

The library has no `default_cell_leakage_power` specification or `cell_leakage_power` specification for the AND2x2 cell.

```
cell (AND2x2) {  
    area : 4;  
    ...  
}
```

In this case, the RC-LP engine uses the `default_leakage_power_density` and the `area` to calculate the leakage power of the AND2x2 cell.

$$\text{cell_leakage_power} = \text{area} \times \text{default_leakage_power_density} = 4 \times 0.5 = 2 \text{ nW}$$

Providing Switching Activity Information

- [Introduction](#) on page 58
- [Sources of Switching Activity Information](#) on page 60
 - [Using Default Switching Activities](#) on page 61
 - [Propagating Net Switching Activities](#) on page 62
 - [Overriding Net Switching Activities](#) on page 62
 - [Determining the Source of the Switching Activity Information](#) on page 63
- [Reading Switching Activity Information from a TCF File](#) on page 64
 - [Reading a TCF File with Modified Clock Frequency](#) on page 65
 - [Reading Multiple TCF Files with Different Weights](#) on page 66
 - [Reading Instance-Based Switching Activities](#) on page 67
- [Reading Switching Activity Information from an SAIF File](#) on page 71
- [Reading Switching Activity Information from a VCD File](#) on page 72
 - [Reading Instance-Based Switching Activities](#) on page 73
 - [Displaying an Activity Profile](#) on page 75
- [Checking System Messages when Reading Switching Activities](#) on page 80

Introduction

As shown in [Power Calculation](#) on page 51, the RC-LP engine uses the probability and toggle rate data to estimate the power components.



Tip

The probability and toggle rate data are also referred to as *net switching activities*.

The RC-LP engine reads the user-defined switching activity information from the following net attributes:

- `lp_asserted_probability` specifies the probability of a net being high.
- `lp_asserted_toggle_rate` specifies the toggle count per toggle rate unit of a net.

[Sources of Switching Activity Information](#) describes how the `lp_asserted_probability` and `lp_asserted_toggle_rate` net attributes can be set.

Note: To retrieve the probability and toggle rate for any net (asserted or not), you can use the following attributes:

```
get_att lp_computed_probability [find . -net net]
get_att lp_computed_toggle_rate [find . -net net]
```

[Figure 4-1](#) on page 59 shows where in the flow you can provide the switching activity information.

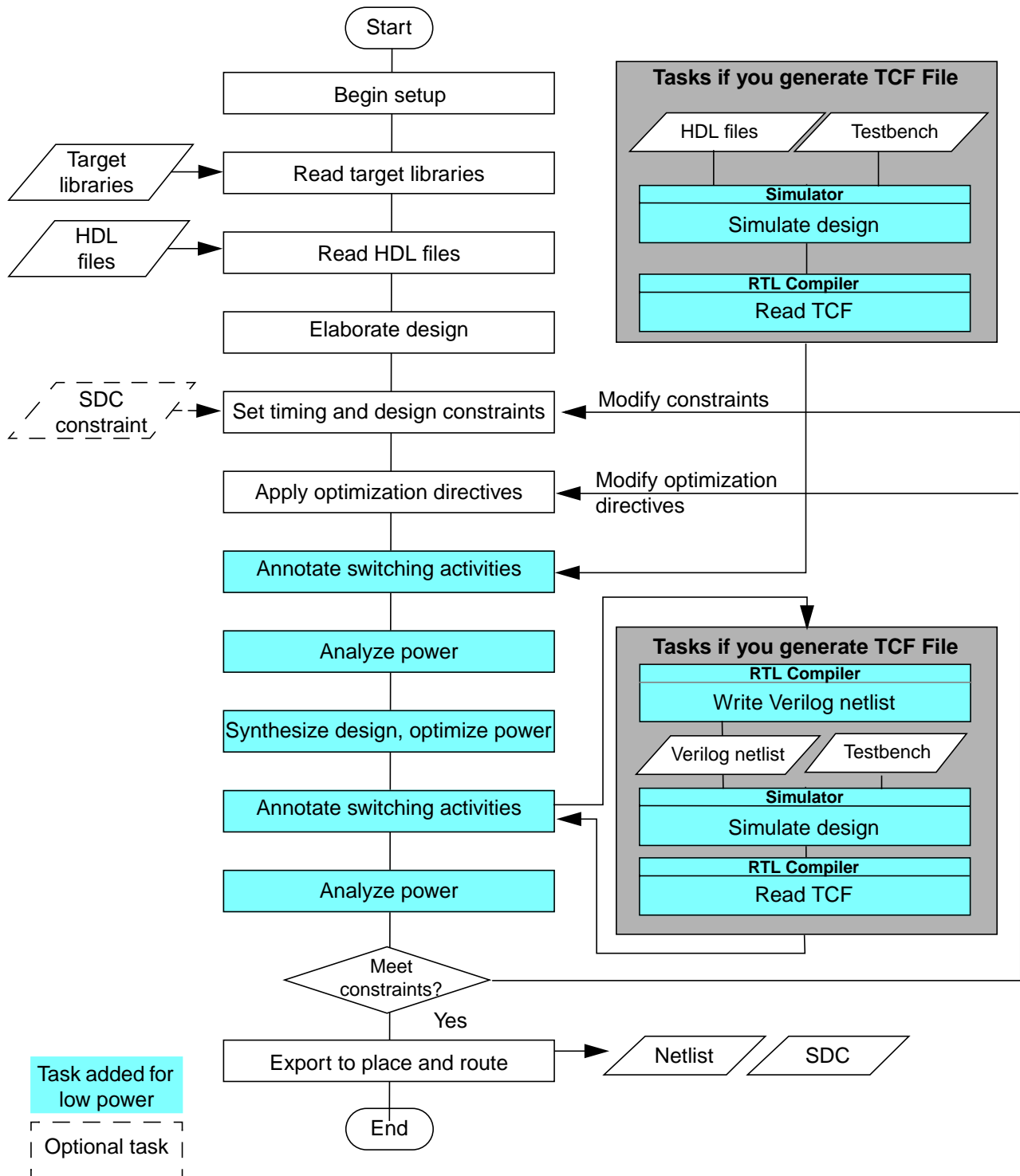
- For a more accurate power optimization, annotate switching activities before synthesis.
- For a more accurate power calculation, annotate switching activities before power analysis.

Note: [Running Simulation to Generate TCF Files in Cadence Toggle Count Format](#) shows several scenarios to generate a TCF file through simulation. The flow in [Figure 4-1](#) applies to the scenarios where you perform simulation with a Verilog netlist.

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

Figure 4-1 Power Analysis in the Top-Down Low Power Synthesis Flow



Sources of Switching Activity Information

The RC-LP engine uses the following order to determine the value of the `lp_asserted_probability` and `lp_asserted_toggle_rate` net attributes.

1. If you provide a Toggle Count Format (TCF) file, a Switching Activity Interchange Format (SAIF) file, or a Value Change Dump (VCD) file, the RC-LP engine reads the net switching activities from this file.

[Reading Switching Activity Information from a TCF File](#) lists the commands you need to use to read in a Toggle Count Format (TCF) file.

[Reading Switching Activity Information from an SAIF File](#) describes how to read in this file.

You can first read in a SAIF file for the design, and then read in a TCF file to update the switching activities for the same design.

[Reading Switching Activity Information from a VCD File](#) describes how to use this file.



Tip

To obtain accurate power analysis results, generate real switching activity data by simulating the design.

TCF, SAIF and VCD files can be generated by simulating the design. [Figure 4-1](#) on page 59 shows the tasks involved when generating a TCF file. Similar tasks apply for generating a SAIF or VCD file.

[Running Simulation to Generate TCF Files](#) explains more about the steps involved in the simulation.

2. If you want to skip simulation, you can set the net switching activities using the following commands:

```
set_attribute lp_asserted_probability float /designs/design/*/nets/net
set_attribute lp_asserted_toggle_rate float /designs/design/*/nets/net
```

3. For *clock* nets, the RC-LP engine derives the net switching activities from the clock definition in the timing constraints.
4. For nodes that have no user-asserted switching activities, the RC-LP engine can either
 - ☐ Use default switching activities (see [Using Default Switching Activities](#))
 - ☐ Propagate switching activities to those nets that have *no* switching information asserted (see [Propagating Net Switching Activities](#))

Using Default Switching Activities

If some nodes have no user-asserted switching activities, and you want a fast power analysis you can choose to apply the default switching activities for the remaining non-asserted nodes.

- To use the default probability and toggle rate values for power analysis, set the following root attribute to `low`:

```
set_attribute lp_power_analysis_effort low /
```

This attribute setting will cause the RC-LP engine to apply the default probability and toggle rate to each net that has no user-asserted switching activities.

- The *tool*/ default signal probability is 0.5, the tool default toggle rate is 0.02, and the default toggle rate unit is per nanosecond.

- To change these *tool*/ defaults set the following attributes:

```
set_attribute lp_default_probability float {design | hierarchical_instance}  
set_attribute lp_default_toggle_rate float design | hierarchical_instance}  
set_attribute lp_toggle_rate_unit toggle_unit /
```

Note: These default attribute settings are not hierarchical. If you set the defaults on a hierarchical instance, the RC-LP engine only applies these specified default switching activities to nets which are driven by leaf instances within this hierarchical instance.

- To take the effect of the clocks in the design into account, you can associate a scaling factor with each clock by setting the following attribute:

```
set_attribute lp_default_toggle_percentage value clock
```

This attribute specifies the multiplication factor to be used with this clock to modify the default toggle rate of any data pin affected by this clock. The toggle rate of a data pin is derived by multiplying this scale factor with the toggle rate of the clock. If multiple clocks are related to the specified data pin, the fastest clock is used.

If a pin is not affected by any clock, the user-defined defaults apply, otherwise the tool defaults are used.

The low effort mode does not return accurate switching activity estimates, but the power analysis runs fast, consumes little memory, and gives a quick estimate on the overall power dissipation of the design.

Propagating Net Switching Activities

If some nodes have no user-asserted switching activities, the RC-LP engine can automatically propagate the switching activities from the nodes in the fanin cone that contain asserted values. If none of the logic nodes in the fanin cone contain asserted values, the RC-LP engine traces back to the primary input(s) and propagates the default switching activities.

- To control the effort used to propagate the switching activities, set the `lp_power_analysis_effort` root attribute as follows:

```
set_attribute lp_power_analysis_effort {medium|high} /
```

Both modes (efforts) provide accurate switching information, run slower and take more memory than the `low` effort. The `high` effort will provide the most accurate information, but at the cost of slower run time and more memory than the `medium` effort.

The default is `medium`, indicating that the switching activities are propagated.

Overriding Net Switching Activities

You can override the current net switching activities at any time using the following commands:

```
set_attribute lp_asserted_probability float /designs/design/*/nets/net
set_attribute lp_asserted_toggle_rate float /designs/design/*/nets/net
```



Tip

The RC-LP engine does not take the `timing_case_logic_value` pin attribute into account when it performs power analysis. For example, the following two sets of commands return the same results:

```
rc:/> dc::set_case_analysis 0 [dc::get_port TEST_MODE]
rc:/> report power
```

and

```
rc:/> dc::set_case_analysis 1 [dc::get_port TEST_MODE]
rc:/> report power
```

The RC-LP engine estimates power based on the average effect, while the timing case is considered as a single event. If you determine that the timing case value has an average effect on the power analysis, you can force the RC-LP engine to use the timing case value by setting the following attributes:

```
set_attribute lp_asserted_probability timing_case_value [get_attr net pin]
set_attribute lp_asserted_toggle_rate 0 [get_attribute net pin]
```

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

Troubleshooting

- If you try to set the `lp_asserted_probability` attribute to a value other than 0 or 1 on a net driven by a constant, the RC-LP engine will report a failure.
- If you try to set the `lp_asserted_toggle_rate` attribute to a value other than 0 on a net driven by a constant, the RC-LP engine will report a failure.

Determining the Source of the Switching Activity Information

- To find the source of the probability information, use the following command:

```
get_att lp_probability_type /designs/design/*/nets/net
```

- To find the source of the toggle rate information, use the following command:

```
get_att lp_toggle_rate_type /designs/design/*/nets/net
```

These commands can return the following values:

asserted	Indicates that the net value is user-specified. You either specified the value through the <code>lp_asserted_probability</code> (<code>lp_asserted_toggle_rate</code>) attribute or in a TCF or SAIF file.
clock	Indicates that the net is a clock net and that the value is derived from the clock waveform. Note: If you asserted the switching activities on the clock net, the user-asserted values will always take precedence and the source for the probability or toggle rate information will be listed as <i>asserted</i> .
computed	Indicates that the net value is computed by propagating the internal switching activities.
constant	Indicates that the net is driven by a constant value. In this case, the value of the <ul style="list-style-type: none">■ <code>lp_computed_probability</code> attribute returns 1 if it is driven by a logic 1, or 0 if it is driven by a logic 0.■ <code>lp_computed_toggle_rate</code> attribute is set to 0.
default	Indicates that the net value is not user-specified, but determined by the value of the <code>lp_default_probability</code> (<code>lp_default_toggle_rate</code>) attribute.

Reading Switching Activity Information from a TCF File

A Cadence® Toggle Count Format (TCF) file includes the toggle count information and the probability of a net or pin to be in the logic 1 state. For more information about the syntax, refer to [TCF Syntax](#).

If you generated a TCF file through simulation, the file contains the switching activity information on all the nets for the specified monitoring scope (see [Running Simulation to Generate TCF Files](#)).

► To read a TCF file:

- ❑ Use the `read_tcf` command to read the toggle count information and probability information of the pins and nets, and the simulation period. This command overwrites any existing information on the pins and nets.

```
read_tcf [-scale scale_factor] file
```

- ❑ Use the `read_tcf -update` command to update the toggle count information and probability information of the pins and nets, and the simulation period.

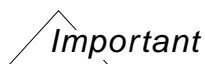
```
read_tcf -update [-scale scale_factor] [-weight weight_factor] file
```

If you generated your TCF file with a different clock frequency than the one used to synthesize the design, refer to [Reading a TCF File with Modified Clock Frequency](#) for more information on how to read in that file.

If you generated a new TCF file for a different time period or for a different testbench than the one used to generate the first TCF file, refer to [Reading Multiple TCF Files with Different Weights](#) for more information on how to read in the second file.

You can read in an incomplete TCF file to annotate the switching activities for a hierarchical instance of a *full* chip design loaded in RTL Compiler. You can also extract the switching activities of an instance in a *full* chip simulation TCF file to annotate the switching activities for the *partial* design loaded in RTL Compiler. Refer to [Reading Instance-Based Switching Activities](#) for more information.

For more information on the accuracy of the generated TCF file, refer to [Impact of Simulation Mode on Generated TCF Files](#).



To get accurate power analysis results, it is imperative to use the same netlist for power analysis that you used to create the TCF file.

Reading a TCF File with Modified Clock Frequency

In case you have *setup violations* during gate-level simulation, you might need to reduce the simulation clock frequency to resolve those violations to generate an accurate TCF file.

If the switching activity information in the TCF file applies to a different clock frequency than the original one, you must indicate this to the RC-LP engine, because it needs the switching activity information for the original clock frequency. By specifying a scaling factor—the ratio of the original clock period over the new clock period—when you read in the TCF file, the RC-LP engine can derive the switching activity information for the original clock frequency.

$$\text{scale_factor} = \frac{\text{original clock period}}{\text{simulation clock period}}$$

The values in the new TCF file would be:

$$\text{new_TCF_value} = \frac{\text{original TCF_value}}{\text{scale_factor}}$$

Example

Assume that in the original testbench, the clock frequency is 100 Mhz (10 ns clock period). During simulation you reduce the clock frequency to 10 Mhz (100 ns clock period) to prevent *setup violations*, and you create a TCF file named `slowed.tcf`.

To read in this TCF file in RTL Compiler, you need to use the following command:

```
rc:/> read_tcf -scale 0.1 slowed.tcf
```

Reading Multiple TCF Files with Different Weights

Assume you already read a TCF file, but you want to read in another TCF file. If for power calculation purposes, you want to put a different weight on the new TCF file, you can do so by specifying a weight factor when you read the new file with the `read_tcf -update` command. The weighted probability and toggle rate values are calculated as follows:

$$prob_w = \frac{(prob_p + w \times prob_n)}{1 + w}$$
$$tr_w = \frac{(tr_p + w \times tr_n)}{1 + w}$$

where

- $prob_w$ and tr_w are the weighted probability and toggle rate

Probability (prob) and toggle rate (tr) is defined as:

$$prob = \frac{time_signal_one}{simulation_period}$$
$$tr = \frac{toggle_count}{simulation_period}$$

- $prob_p$ and tr_p are the previous probability and toggle rate

Note: If the RC-LP engine did not propagate any switching activities between reading the two TCF files, the previous probability and toggle rate correspond to the values in the first TCF file.

- $prob_n$ and tr_n are the probability and toggle count in the new TCF file
- w is the relative weight you assign to the switching activities in the new TCF file with respect to the current switching activities when you read in the new TCF file using the `read_tcf -update` command

Note: You can specify any non-negative value for the weight.

The default weight is 1.0. To give equal weight to old and new values of probability and toggle count, set the weight to 1.0.

Example

```
rc:/> read_tcf test_bench_1.tcf
rc:/> read_tcf -update -weight 2 test_bench_2.tcf
```

Reading Instance-Based Switching Activities

The RC-LP engine can read switching activities

- From a partial TCF file for an instance of a *full* chip loaded in RTL Compiler ([Example 4-1](#) on page 67)
- From a full chip TCF file for a *partial* design loaded in RTL Compiler ([Example 4-2](#) on page 69), where the partial design is an instance in the full chip in the TCF file

In both cases, you use the `-instance` option to refer to either the instance name in the design loaded in RTL Compiler, or the instance name in the TCF file.

Example 4-1 Reading a TCF File for an Instance of a Design Loaded in RTL Compiler

[Figure 4-2](#) on page 67 shows the netlist of design `mult_bit_muxed_add` that is loaded in RTL Compiler.

[Figure 4-3](#) on page 68 shows the TCF file. The TCF file does not contain any instance name.

[Figure 4-4](#) on page 68 shows the `read_tcf` command and summary.

When you read in the TCF file with the `-instance` option, the RC-LP engine searches for the specified instance `ma0` in the loaded design. When it finds the instance in the design, it applies the content of the TCF file to this instance.

Figure 4-2 Netlist of Full Design Loaded in RTL Compiler

```
module muxed_add(ck, en, rst, a, b, c, d, s, y);
    input ck, en, rst, a, b, c, d, s;
    output y;
    wire ck, en, rst, a, b, c, d, s;
    wire y;
    wire n_0, n_1, n_3, n_4, n_5, n_6, n_7, n_8;
    wire n_9, n_10, n_11, n_12, n_13, y_21;
    DFFNRX4 y_reg(.RN (n_0), .CKN (ck), .D (n_13), .Q (), .QN (y_21));
    NAND3X2 g2(.A (n_11), .B (n_12), .C (n_7), .Y (n_13));
    NAND2X2 g3(.A (n_9), .B (n_8), .Y (n_12));
    NAND2X4 g4(.A (n_6), .B (n_10), .Y (n_11));
    AOI21X4 g5(.A0 (d), .A1 (b), .B0 (n_5), .Y (n_10));
    MXI2X4 g6(.S0 (c), .B (a), .A (n_1), .Y (n_9));
    INVX2 g7(.A (n_4), .Y (n_8));
    NAND2BXL g9(.AN (y_21), .B (n_3), .Y (n_7));
    NOR2BX2 g10(.AN (en), .B (s), .Y (n_6));
    NOR2X4 g11(.A (d), .B (b), .Y (n_5));
    NAND2X2 g8(.A (en), .B (s), .Y (n_4));
    CLKINVX3 g12(.A (en), .Y (n_3));
    INVX2 g13(.A (y_21), .Y (y));
    CLKINVX4 g14(.A (a), .Y (n_1));
    INVXL g15(.A (rst), .Y (n_0));
endmodule
module muxed_add_7(ck, en, rst, a, b, c, d, s, y);
    ...
endmodule
```

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

```
module muxed_add_17(ck, en, rst, a, b, c, d, s, y);
...
endmodule
module mult_bit_muxed_add(ck, en, rst, a, b, c, d, s, y);
    input ck, en, rst, s;
    input [2:0] a, b, c, d;
    output [2:0] y;
    wire ck, en, rst, s;
    wire [2:0] a, b, c, d;
    wire [2:0] y;
    muxed_add ma0(.ck (ck), .en (en), .rst (rst), .a (a[0]), .b (b[0]),
    .c (c[0]), .d (d[0]), .s (s), .y (y[0]));
    muxed_add_7 ma1(.ck (ck), .en (en), .rst (rst), .a (a[1]), .b (b[1]),
    .c (c[1]), .d (d[1]), .s (s), .y (y[1]));
    muxed_add_17 ma2(.ck (ck), .en (en), .rst (rst), .a (a[2]), .b
    (b[2]), .c (c[2]), .d (d[2]), .s (s), .y (y[2]));
endmodule
```

Figure 4-3 Partial TCF File for Instance ma0

```
tcffile () {
    tcfversion      :      "1.0";
    duration        :      "1.000000e+03"; /* would grab the duration*/
    unit            :      "ns";
    instance () {
        pin() {
            "y_reg/CKN"      :      "0.6 40";
            "g12/A" :      "0.4 10";
            "g15/A" :      "0.2 2";
            "g14/A" :      "0.4 10";
            "g11/B" :      "0.4 10";
            "g6/S0" :      "0.4 10";
            "g5/A0" :      "0.4 10";
            "g8/B" :      "0.4 10";
        }
    }
}
```

Figure 4-4 read_tcf Summary

```
rc:/> read_tcf -instance [find / -inst ma0] ma0_pin.tcf
Reading ma0_pin.tcf
...
90.0 % done
Nets/Pins asserted in TCF file : 8
Total Nets/Pins in TCF file   : 8
-----
Asserted Primary inputs in design      : 8 (50.00%)
Total Primary inputs in design         : 16 (100.00%)
-----
Asserted sequential outputs            : 3 (50.00%)
Total sequential outputs                : 6 (100.00%)
-----
Total nets in design                   : 88 (100.00%)
Nets asserted                          : 24 (27.27%)
Clock nets                            : 0 (0.00)
Constant nets                         : 0 (0.00)
Net does not have TCF asserted         : 64 (72.73%)
-----
```

Example 4-2 Extracting Switching Activities of an Instance from a Full Chip TCF File

Figure 4-5 on page 69 shows the netlist of the partial design loaded in RTL Compiler. The name of the design is `mid`.

Figure 4-6 on page 70 shows the TCF file for design `top` of which `mid` is an instance. The TCF file is a hierarchical TCF file.

Figure 4-7 on page 70 shows the `read_tcf` command and summary.

When you read in the TCF file with the `-instance` option, the RC-LP engine first searches for the specified instance `mid` in the loaded design. When it cannot find an *instance* of that name in the design, it searches for that instance name in the TCF file and applies the switching activities of that instance to the design loaded in RTL Compiler.

Figure 4-5 Netlist of Partial Design Loaded in RTL Compiler

```
module bot(do, di, load, clk);
    output [6:0] do;
    input [6:0] di;
    input load;
    input clk;

    AND2X1 i_2_LPS_CG_GATING_1(.A(clk), .B(i_2_n_21), .Y(n_0));
    TLATNX1 i_2_LPS_CG_LATCH(.D(load), .GN(clk), .Q(i_2_n_21));
    DFFHQX1 \$$xx$do_reg_6[1] (.D(di[6]), .CK(n_0), .Q(do[6]));
    DFFHQX1 do_reg_5(.D(di[5]), .CK(n_0), .Q(do[5]));
    DFFHQX1 do_reg_4 (.D(di[4]), .CK(n_0), .Q(do[4]));
    DFFHQX1 do_reg_3(.D(di[3]), .CK(n_0), .Q(do[3]));
    DFFHQX1 do_reg_2(.D(di[2]), .CK(n_0), .Q(do[2]));
    DFFHQX1 do_reg_1(.D(di[1]), .CK(n_0), .Q(do[1]));
    DFFHQX1 do_reg_0(.D(di[0]), .CK(n_0), .Q(do[0]));
endmodule

module mid(odata, idata, irin, clk);
    output [6:0] odata;
    input [6:0] idata;
    input [1:5] irin;
    input clk;

    wire [6:0] hdata;
    wire [1:5] ireg;

    DFFHQX1 ireg_reg_5(.D(irin[5]), .CK(clk), .Q(ireg[5]));
    DFFHQX1 ireg_reg_4(.D(irin[4]), .CK(clk), .Q(ireg[4]));
    DFFHQX1 ireg_reg_3(.D(irin[3]), .CK(clk), .Q(ireg[3]));
    DFFHQX1 ireg_reg_2(.D(irin[2]), .CK(clk), .Q(ireg[2]));
    DFFHQX1 ireg_reg_1(.D(irin[1]), .CK(clk), .Q(ireg[1]));
    bot boti (.do(hdata), .di(idata), .load(ireg[1]), .clk(clk));
endmodule
```

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

Figure 4-6 Full Chip TCF File

```
tcf file () {
  tcfversion      :      "1.0";
  duration        :      "7.200000e+03";
  unit            :      "ns";
  instance ("top_sim") {
    instance ("top") {
      instance ("mid") {
        instance ("boti") {
          instance ("do_reg_5") {
            pin () {
              "Q"      :      "0.000000  1";
              "D"      :      "1.000000  0";
            }
          }
          instance ("do_reg_4") {
            pin () {
              "Q"      :      "0.000000  1";
              "D"      :      "1.000000  0";
            }
          }
        }
      }
    }
  }
}
```

Figure 4-7 read_tcf Summary

```
rc:/> read_tcf -instance mid instance.tcf
Reading instance.tcf
0.0 % done
10.0 % done
20.0 % done
30.0 % done
40.0 % done
50.0 % done
60.0 % done
70.0 % done
80.0 % done
90.0 % done
Nets/Pins asserted in TCF file : 4
Total Nets/Pins in TCF file   : 4
-----
Asserted Primary inputs in design      : 2 (22.22%)
Total Primary inputs in design         : 9 (100.00%)
-----
Asserted sequential outputs            : 3 (30.00%)
Total sequential outputs               : 10 (100.00%)
-----
Total nets in design                   : 36 (100.00%)
Nets asserted                          : 6 (16.67%)
Clock nets                             : 0 (0.00)
Constant nets                          : 0 (0.00)
Net does not have TCF asserted         : 30 (83.33%)
-----
```

Reading Switching Activity Information from an SAIF File

The SAIF file provides detailed information about the switching behavior of nets and ports, which leads to more accurate power estimation.

Use the `read_saif` command to read switching activity information from Synopsys Switching Activity Interchange Format (SAIF) and to convert it internally to the Toggle Count Format (TCF) for power estimation.

```
read_saif [-scale scale_factor] [-update [-weight weight_factor]]  
          [-instance instance] file
```

Like for TCF files, you can read in an incomplete SAIF file to annotate the switching activities for a hierarchical instance of a *full* chip design loaded in RTL Compiler. You can also extract the switching activities of an instance in a *full* chip simulation SAIF file to annotate the switching activities of the *partial* design loaded in RTL Compiler.

Note: The RC-LP engine reads in the *back-annotation* file generated by the Synopsys simulator which contains information about the observed switching activity for those elements monitored during simulation.

Reading Switching Activity Information from a VCD File

A Value Change Dump (VCD) file contains information about value changes on selected signals in your design. A VCD file is typically created by a simulator and requires adding VCD system tasks to the Verilog or VHDL source file.

Use the `read_vcd` command to read switching activity information from a VCD file.

```
read_vcd
  [-static
  | -activity_profile [-time_window time] [-simvision] [-write_sst2 file] ]
  [-start_time start_monitoring_time] [-end_time end_monitoring_time]
  [-module {design|subdesign}] [-vcd_module module]
  vcd_file
```

Note: If you specify the `read_vcd` command without any options, a static power analysis is performed by default.

Whether the switching activities are annotated to the design or not, depends on the option you specify with the `read_vcd` command:

- `-static` causes the switching activities to be annotated to the design.

In this case, you can perform regular power analysis using the `report power` command. For more information see [Chapter 5, "Power Analysis."](#)

- `-activity_profile` builds a profile of the activities for the set scope without annotating the switching activities to the design.

In this case, you can

- Use the `-simvision` option to invoke SimVision and display the activity profile

For more information, see [Displaying an Activity Profile](#).

- Use the `-write_sst2` option to generate SST2 database files to view the data in other waveform viewers

See [Setting the Scope](#) for more information on how to set the scope.

Note: The time values for the `-start_time`, `-end_time`, and `-time_window` options must be specified in picoseconds.

Reading Instance-Based Switching Activities

Like for TCF files, you can read in an incomplete VCD file to annotate the switching activities for a hierarchical instance of a *full* chip design loaded in RTL Compiler. You can also extract the switching activities of an instance in a *full* chip simulation VCD file to annotate the switching activities of the *partial* design loaded in RTL Compiler.

Example 4-3 Reading a VCD File for an Instance of a Design Loaded in RTL Compiler

In this example, the design is a super-hierarchy of the VCD. Design `top` is loaded in RTL Compiler, while the VCD file contains the information for module `mid2` of design `top`.

Hierarchy of Full Design Loaded in RTL Compiler

```
top
|__ mid1
|   |__ FF0 (leaf level)
|__ mid2
|   |__ bot1
|       |__ FF1 (leaf level)
|   |__ bot2
|       |__ FF2 (leaf level)
```

Partial VCD

```
mid2
|__ bot1
|   |__ FF1 (leaf level)
|__ bot2
|   |__ FF2 (leaf level)
```

Command Usage

You need to use the `-module` option to specify the name of the subdesign in the RTL Compiler hierarchy to which the parsed VCD hierarchy (specified through the `-vcd_module` option) corresponds.

```
read_vcd -module mid2 -vcd_module mid2 ...
```

Example 4-4 Extracting Switching Activities of an Instance from a Full Chip VCD File

In this case, the design is a sub-hierarchy of the VCD.

The VCD file contains the information for design `top`, while only part of its hierarchy (module `RC_mid2`) is loaded in the RTL Compiler hierarchy.

Hierarchy of Partial Design Loaded in RTL Compiler

```
RC_mid2
|__ bot1
|   |__ FF1 (leaf level)
|__ bot2
|   |__ FF2 (leaf level)
```

Full VCD

```
top
|__ mid1
|   |__ FF0 (leaf level)
|__ mid2
|   |__ bot1
|       |__ FF1 (leaf level)
|   |__ bot2
|       |__ FF2 (leaf level)
```

Command Usage

You need to use the `-vcd_module` option to specify the name of module in the VCD file to which the design loaded in the RTL Compiler hierarchy corresponds.

```
read_vcd -vcd_module mid2 ...
```

Note: If you do not specify the `-module` option, the module in the RTL Compiler hierarchy defaults to the design, which is module `RC_mid2`.

Displaying an Activity Profile

- Use the `-activity_profile` option when reading in the VCD file to build a profile of the activities for the selected scope.

The activities refer to the toggle counts of the nets inside the scope. The activity of a hierarchical instance is the sum of the activities of all the nets in that instance as well as activities of all hierarchical instances in the module of that instance.

Using a waveform viewer you can look during which time window the peak activity occurs.

Reading a VCD file for a full chip can be time consuming. You can control the runtime by

- [Setting the Scope](#)
- [Choosing Your Time Window](#)
- [Choosing the Monitoring Time](#)

Setting the Scope

If you are time-limited, you should limit the scope to only read part of the VCD file.

- To limit the scope to parts of the design hierarchy, set the following instance attribute:
`set_attribute lp_dynamic_analysis_scope true instance_list`

If you do not set this attribute, activity profiling is done for the scope parsed in the VCD file, which is determined by the `-vcd_module` option.

Note: The activity profile of all user-specified hierarchies can be displayed in a waveform viewer.

Choosing Your Time Window

The selection of the time window also affects the runtime to read the VCD file, build the activity profile and generate the database for the waveform viewer.

Your time window should not be any smaller than your smallest clock cycle.

If you omit the `-time_window` option, the RC-LP engine calculates the window based on the start and end monitoring times and the value of the `lp_power_analysis_effort` root attribute.

- For a *low* effort, the time window is determined as one tenth of the monitoring time.
- For a *medium* effort, the time window is determined as one fiftieth of the monitoring time.
- For a *high* effort, the time window is determined as one hundredth of the monitoring time.

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

Note: In this case, you start with reading in the VCD file using the information of the entire simulation time.

Possible Approach

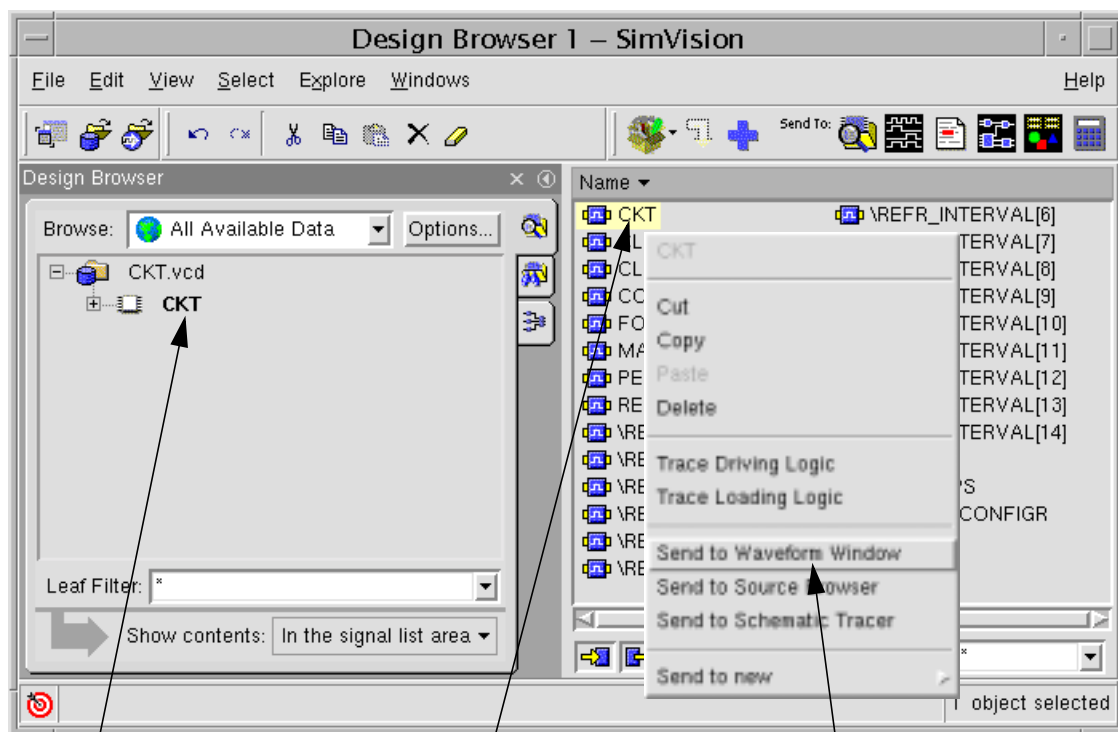
1. Start with a larger time window—a certain percentage of the monitoring time. A quick check in the waveform viewer can give you an idea of the period of interest. For example:

```
read_vcd -activity_profile -time_window 1800500 -vcd_module $mod my.vcd  
-simvision
```

This command launches SimVision. Follow the directions in Figure 4-8 to display the activity profile in the Waveform window.

Note: The width of the time window affects the resolution. If the time window is too large, it can mask the real peak activities. If it is too small, it affects the runtime.

Figure 4-8 Steps in Design Browser Window to Display Activity Profile



1. Click left on design name to display the scope and all nets in the scope in right pane.

2. Click right on the design name to display popup menu.

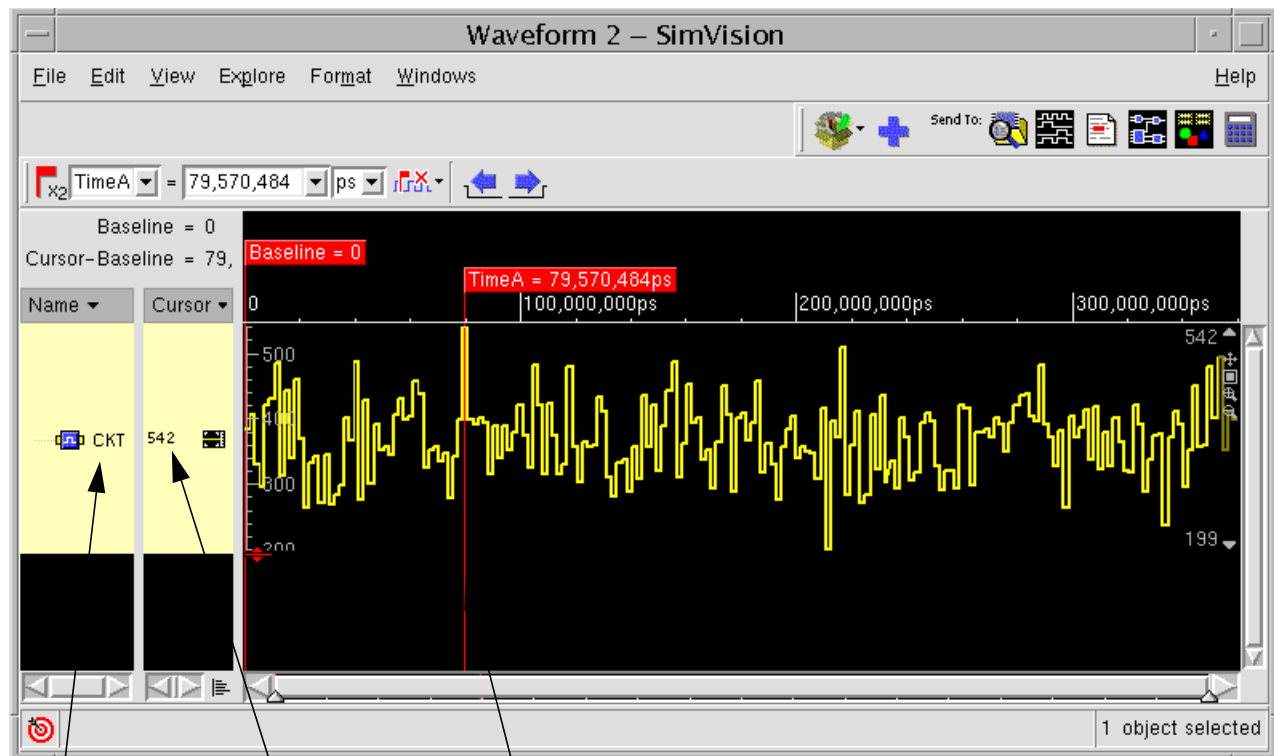
3. Click right on *Send to Waveform Window*.

Figure 4-9 shows the Waveform Window with the activity profile for the design.

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

Figure 4-9 Activity Profile for the Design over Entire Simulation Time



Design name

Toggle count for
current cursor
position

Cursor position

2. Move the cursor to find the time window with highest activity.
3. Read in the VCD file again using the `-start_time` and `-end_time` options to focus on the window of interest and adjust the time window in function of the monitored time.

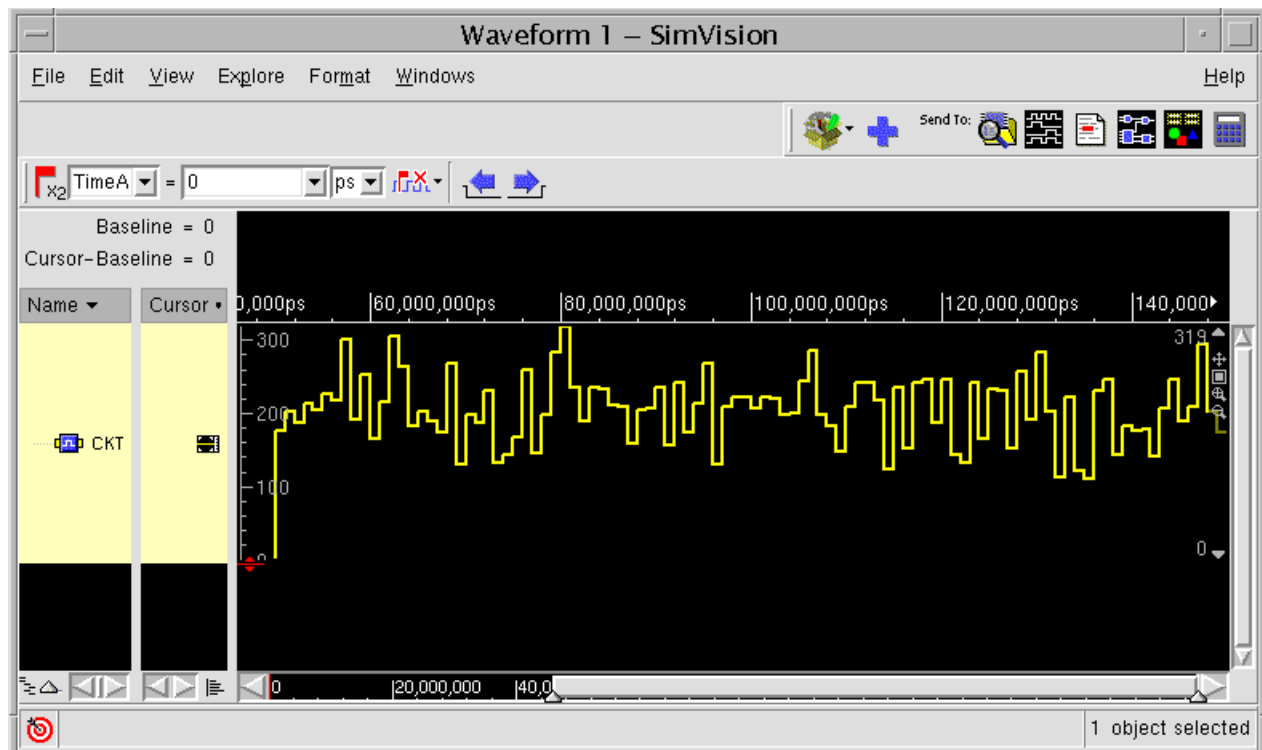
```
read_vcd -activity_profile -time_window 1000000 -start_time 50000000 \
-end_time 150000000 -vcd_module $mod my.vcd -simvision
```

The result is shown in [Figure 4-10](#) on page 78.

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

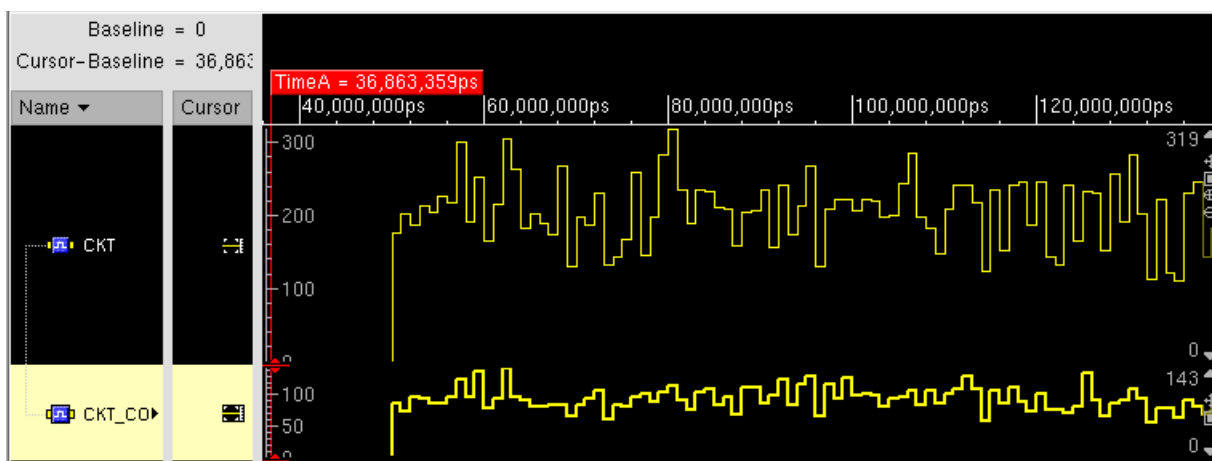
Figure 4-10 Activity Profile for the Design over Specific Time Window



4. Explore the hierarchy and add the activity profile for any hierarchical instance of interest to the Waveform window as shown in Figure 4-11.

Exploring the hierarchy helps pinpoint the hierarchical instance that causes the highest peak activity and that will consume the most power. The activity profile of combinational and sequential instances shows the activities of their pins.

Figure 4-11 Activity Profile for the Design and Hierarchical Instance



Choosing the Monitoring Time

The selection of the monitoring time also affects the runtime to read the VCD file, build the activity profile and generate the database for the waveform viewer.

Possible approach

1. Divide the entire simulation time in equal time intervals and use the same time window to build the activity profile for each time interval.

```
read_vcd -activity_profile -time_window window1 -start_time t1_start \  
-end_time t1_end -vcd_module $mod my.vcd -simvision  
...  
read_vcd -activity_profile -time_window window1 -start_time tn_start \  
-end_time tn_end -vcd_module $mod my.vcd -simvision
```

Note: Each `read_vcd` command opens a separate Simvision session.



Tip

For best turn around time, you can run several RTL Compiler sessions in parallel.

2. Compare the activity profiles in all time intervals (from $t1$ to tn) in the corresponding Waveform windows and identify the time interval with the peak activities.
3. Read in the VCD file for the time interval with the peak activities using a smaller time window in that time range to narrow down on the region.

```
read_vcd -activity_profile -time_window window2 -start_time tc_start \  
-end_time tc_end -vcd_module $mod my.vcd -simvision
```

Checking System Messages when Reading Switching Activities

It is important to check the system messages given when the TCF or SAIF file is read in. Similar messages are given when you read in a VCD file for static power analysis.

For example, the output of reading a TCF file might look like:

```
Reading file : flat.tcf
0.0 % done
10.0 % done
20.0 % done
30.0 % done
40.0 % done
50.0 % done
60.0 % done
70.0 % done
80.0 % done
90.0 % done
Nets/Pins asserted in TCF file : 91
Total Nets/Pins in TCF file   : 91
-----
Asserted Primary inputs in design      : 13 (100.00%)
Total connected primary inputs in design : 13 (100.00%)
-----
Asserted sequential outputs            : 32 (100.00%)
Total connected sequential outputs      : 32 (100.00%)
-----
Total nets in design                   : 151 (100.00%)
Nets asserted                          : 151 (100.00%)
Clock nets                             : 0 (0.00)
Constant nets                          : 0 (0.00)
Nets with no assertions                 : 0 (0.00%)
-----
```

The `read_tcf`, `read_saif`, and `read_vcd -static` summary reports

- The number of pins or nets that were asserted in the TCF or SAIF file
Note: This information is not reported by the `read_vcd -static` command
- The total number of primary inputs and sequential outputs that are *connected* in the design.
- The number of primary inputs, sequential outputs, and nets that are *asserted* in the design loaded in RTL Compiler.

Primary input and sequential output coverage are a good indicator of the assertion coverage if the TCF, SAIF or VCD file were generated using RTL simulation.
- *Nets asserted* refer to nets with asserted switching activities (probability and toggle rate).

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

- *Constant nets* refer to nets whose driver is either a constant object 0 or 1.
- *Clock nets* refer to nets that were defined as clocks in the timing (or SDC) constraints and that have no user-defined assertions.

Note: The `read_vcd` command does not report clock nets.

- *Nets with no assertions* refer to nets for which no switching activities were asserted. This can include clock nets.

Troubleshooting



Tip

You can read the TCF or SAIF files with the `-verbose` option if you want to print a message for each net that is asserted.

Number of Nets Asserted is Larger than Number of Nets/Pins Asserted in TCF File

```
Nets/Pins asserted in TCF file : 4
Nets asserted                  : 6 (16.67%)
```

The number of asserted nets can be larger than the number of pins or nets asserted reported in the first line of the summary report. This can happen if some nets were previously asserted before reading in the TCF/SAIF file.

Number of Nets Asserted in TCF file is Smaller than Total Number of Nets in TCF File

```
Nets/Pins asserted in TCF file : 0
Total Nets/Pins in TCF file    : 1
```

This could indicate a problem.

Check if a warning was reported earlier in the log file:

```
Warning : Path not found. [TCF-2]
         : Line : 12 Net/Pin: /designs/mid/mid2/mid3i/i_15/XYZ
```

The pin referred to in the TCF file could not be found in the design loaded in RTL Compiler.

Number of Reported Clock Nets is zero.

```
Clock nets                  : 0 (0.00)
```

For *clock* nets, the RC-LP engine derives the net switching activities from the clock definition in the timing constraints.

Low Power in Encounter RTL Compiler

Providing Switching Activity Information

If you did not specify any clock definitions, the RC-LP engine cannot distinguish a clock net from a regular net and the clock net will be reported as Net does not have TCF asserted.

If you defined the clock, but you asserted the switching activities on the clock net, the user-asserted values will always take precedence and the clock net will be listed as asserted.

Power Analysis

- [Introduction](#) on page 84
- Tasks
 - [RTL Power Analysis](#)
 - [Reporting Clock Tree Power](#) on page 89
 - [Reporting on All Power Components](#) on page 92
 - [Reporting Power for the Design](#) on page 92
 - [Limiting the Number of Hierarchy Levels Shown](#) on page 93
 - [Reporting Power for Specific Instances](#) on page 94
 - [Reporting the Power of the Leaf Instances](#) on page 93
 - [Sorting According to One Power Component](#) on page 94
 - [Net-based Power Reports](#) on page 96
 - [Showing User-Asserted Information](#) on page 96
 - [Reporting Power on Leaf Instances](#) on page 97
 - [Reporting Power Consumption per Used Cell Types](#) on page 98
 - [Reporting on Specific Power Components for Specific Objects](#) on page 99
- [Troubleshooting](#) on page 102

Introduction

As shown in [Chapter 4, “Providing Switching Activity Information,”](#) the RC-LP engine uses the net switching activities to estimate the power components. [Running Simulation to Generate TCF Files](#) in the *Toggle Count Format Reference* shows you how to get the most accurate switching activity information of each net of the design through simulation.

[Figure 5-1](#) on page 85 shows where in the flow you can annotate switching activities and analyze the power.

- You can analyze power after elaboration.

Power analysis after elaboration is mainly intended to give you an estimation of which block consumes the most power. You can generate a regular power report or you can perform [RTL Power Analysis](#) which cross-references the power consumed by the instances to the corresponding line in the RTL files.

- You can analyze power after synthesis.

Power analysis after synthesis gives you the most accurate results.

If the results after synthesis are not satisfactory, you can

- ☐ Run a more accurate power analysis by providing more accurate switching activities
- ☐ Update the power constraints, reoptimize the design, and issue a new power report

In both cases, after elaboration and after synthesis, the RC-LP engine allows

- [Reporting Clock Tree Power](#)
- [Reporting on All Power Components](#)

Note: You can also analyze all power components module by module.

This method of reporting was specifically tuned for [RTL Power Analysis](#).

- [Reporting Power Consumption per Used Cell Types](#)

Shows how the power may distributed among combinational and sequential logic.

- [Reporting on Specific Power Components for Specific Objects](#)

The output power is given in the units specified in the `lp_power_unit` attribute. The default power unit is nW.

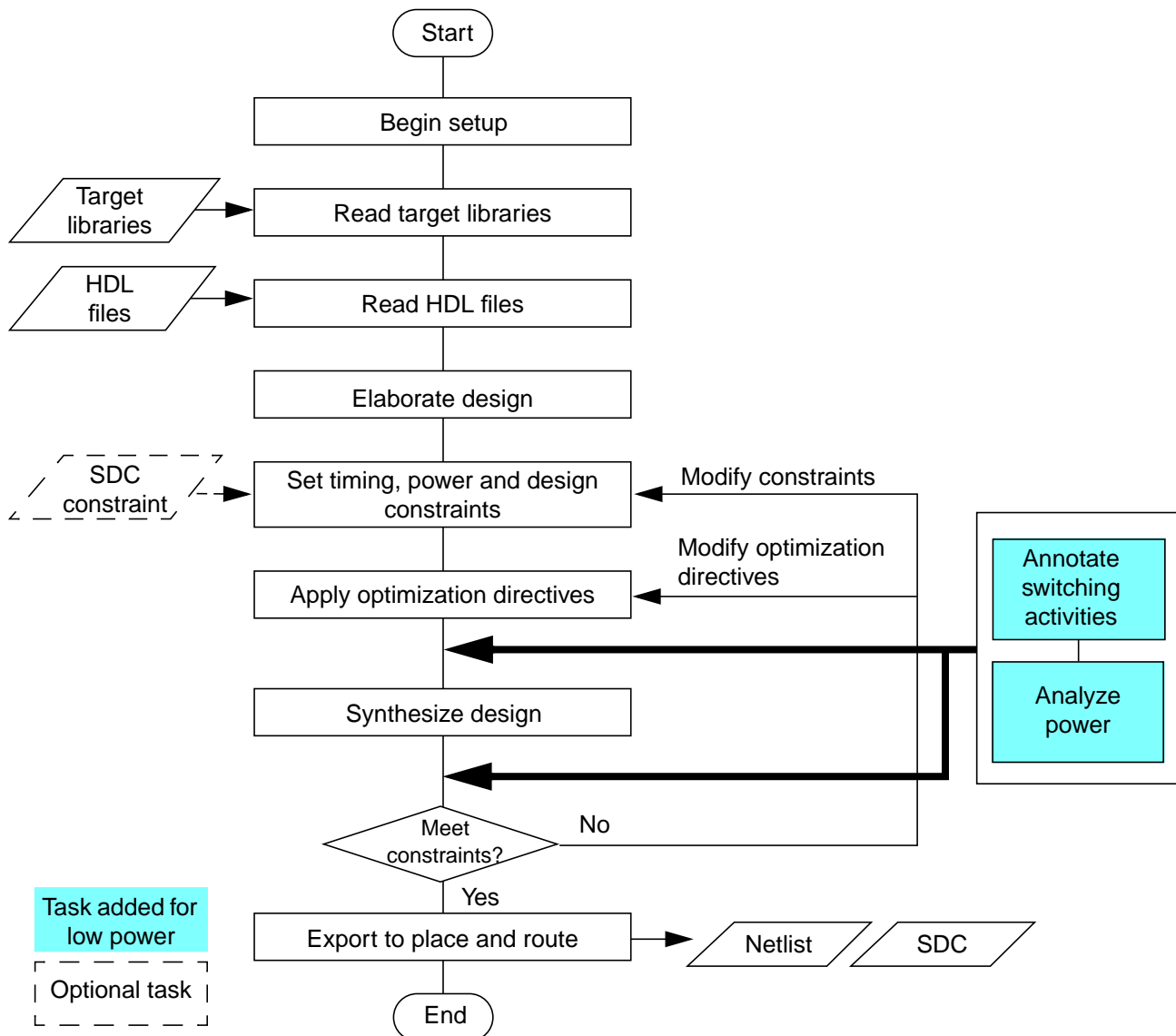
Low Power in Encounter RTL Compiler

Power Analysis

Important

For designs using PSO methodology, the RC-LP engine can perform power domain-aware power analysis. See [Estimate RTL Power](#) and [Analyze Power](#) for more information.

Figure 5-1 Power Analysis in the Top-Down Low Power Synthesis Flow



RTL Power Analysis

RTL power analysis refers to power analysis right after elaboration which allows you to cross-reference the power consumed by the instances to the corresponding line in the RTL files. In this case, the RC-LP engine creates power models for each generic (unmapped) gate based on the given technology libraries. Using the switching activity propagation on the generic netlist, the RTL power analysis engine can give a good prediction of the power consumption after full synthesis and can help you to decide on your optimization strategy.

- To enable RTL power analysis, set the following root attribute before elaborating the design:

```
set_attribute hdl_track_filename_row_col true /
```

This attribute setting enables to cross-reference the power consumed by the instances to the corresponding line in the RTL files.

- To report the power, use the `report power` command with the `-rtl` option:

```
report power -rtl [-detail] [-flat ] [> file]
```

where

`-detail` adds an abbreviated version of the RTL line and a list of the instances that correspond to that RTL line.

`-flat` reports power information for all modules in the current hierarchy. If not specified, only power information for the top-level module in the current hierarchy is shown.

[Example 5-1](#) on page 87 shows a script template for RTL power analysis.

[Example 5-2](#) on page 87 shows a detailed hierarchical report. This report shows

- The net power for each of the primary inputs.
- The power information for the five leaf instances in design `m1` and the total power for hierarchical instance `m2`.

For each *instance* you find:

- ❑ Internal Power—the total internal cell power for the instance (in nW).
- ❑ Leakage Power—the total leakage power for the instance (in nW).
- ❑ Net Power—the net power (in nW) of all nets in the instance.
- ❑ A cross-reference to the corresponding line in the RTL files

[Example 5-3](#) on page 88 shows the detailed flat power report. Note that the last five lines in the report correspond to the last five lines in the hierarchical report. The difference is the flat report reports all leaf instances in `m2`.

Low Power in Encounter RTL Compiler

Power Analysis

Example 5-1 Script for RTL Power Analysis

```
set_attribute lib_search_path ...
set_attribute hdl_search_path ..

# specify the target libraries
set_attribute library library_list /

# enable cross-referencing to RTL files
set_attribute hdl_track_filename_row_col true /

read_hdl design.v
elaborate

# specify timing and design constraints

# run simulation on RTL
# annotate switching activities
read_tcf tcf_file_before_synthesis

#RTL power analysis
report power -rtl
```

Example 5-2 Detailed Hierarchical RTL Power Analysis Report

```
rc:/> report power -rtl -detail
```

```
=====
...
=====
```

Design	Leakage Power(nW)	Internal Power(nW)	Net Power(nW)
m1	1027.349	573618.838	117901.170

Primary Input	Leakage Power(nW)	Internal Power(nW)	Net Power(nW)
clk_ctrl1	0.000	0.000	159.797
clk2	0.000	0.000	16446.240
clk3	0.000	0.000	7698.240
clk1	0.000	0.000	18662.400
tp2	0.000	0.000	143.467
in_1[29]	0.000	0.000	132.970
...			
in_1[0]	0.000	0.000	132.970
en_1[9]	0.000	0.000	454.896
...			
en_1[0]	0.000	0.000	454.896

File	Row	RTL Line	Instances	Leakage Power(nW)	Internal Power(nW)	Net Power(nW)
hier.v	195	m2 m2(..lk1 (clk1), m2		921.353	505488.279	50068.449
hier.v	202	CLKINV..4_clk2_0)); g127		2.833	3514.987	10920.420
hier.v	203	EDFFX1..n_1[9]), .Q o_m..reg_0		33.028	20550.127	116.640
hier.v	205	EDFFX1..n_1[9]), .Q o_m..reg_1		33.641	20510.617	87.480
hier.v	207	EDFFX1..n_1[9]), .Q o_m..reg_2		33.660	20589.636	145.800
hier.v	209	CLKINV.. .Y (n_0)); g132		2.833	2965.192	4914.189

Low Power in Encounter RTL Compiler

Power Analysis

Example 5-3 Detailed Flat RTL Power Analysis Report

```
rc:/> report power -rtl -detail -flat
```

```
=====
```

...	=====					
Design		Leakage Power(nW)	Internal Power(nW)	Net Power(nW)		
m1		1027.349	573618.838	117901.170		

```
-----
```

Primary Input		Leakage Power(nW)	Internal Power(nW)	Net Power(nW)		
clk_ctrl		0.000	0.000	159.797		
clk2		0.000	0.000	16446.240		
clk3		0.000	0.000	7698.240		
clk1		0.000	0.000	18662.400		
tp2		0.000	0.000	143.467		
in_1[29]		0.000	0.000	132.970		
...						
in_1[0]		0.000	0.000	132.970		
en_1[9]		0.000	0.000	454.896		
...						
en_1[0]		0.000	0.000	454.896		

```
-----
```

File	Row	RTL Line	Instances	Leakage Power(nW)	Internal Power(nW)	Net Power(nW)
hier.v	9	MX2X1 .. (o_m6_0)); g14		12.977	12281.589	35207.055
hier.v	23	EDFFX1..n_5[0]), .Q o_m.._reg_0		33.714	20389.659	116.640
hier.v	25	EDFFX1..n_5[0]), .Q o_m.._reg_1		31.673	20389.659	116.640
hier.v	27	EDFFX1..n_5[0]), .Q o_m.._reg_2		33.440	20448.924	160.380
hier.v	29	CLKINV.. .Y (n_0)); g58		2.844	4056.267	4874.823
...						
...						
hier.v	165	EDFFX1..n_2[7]), .Q o_m.._reg_2		33.520	20490.862	72.900
hier.v	167	EDFFX1..n_2[7]), .Q o_m.._reg_1		33.647	20530.372	102.060
hier.v	169	CLKINV.. .Y (n_0)); g150		2.833	2965.192	4914.189
hier.v	170	EDFFX1..n_2[8]), .Q o_m.._reg_0		33.634	21854.837	72.900
hier.v	172	EDFFX1..n_2[8]), .Q o_m.._reg_1		33.517	21953.612	145.800
hier.v	174	EDFFX1..n_2[8]), .Q o_m.._reg_2		33.719	21835.082	58.320
hier.v	202	CLKINV..4_clk2_0)); g127		2.833	3514.987	10920.420
hier.v	203	EDFFX1..n_1[9]), .Q o_m.._reg_0		33.028	20550.127	116.640
hier.v	205	EDFFX1..n_1[9]), .Q o_m.._reg_1		33.641	20510.617	87.480
hier.v	207	EDFFX1..n_1[9]), .Q o_m.._reg_2		33.660	20589.636	145.800
hier.v	209	CLKINV.. .Y (n_0)); g132		2.833	2965.192	4914.189

```
-----
```


Reporting Clock Tree Power

Clock tree power analysis estimates the power dissipated by the clock tree implemented by the physical implementation tool.

Note: The RC-LP engine only supports estimation for balanced trees. Clock structures such as meshes are not supported.

- To estimate the power in the clock tree, use the `report power` command with the `-clock_tree` option:

```
report power -clock_tree [clock]... -buffers libcell_list  
-leaf_max_fanout integer [-width float -height float] [> file]
```

where

- ❑ `clock` is a clock object. If no clock is specified, the power is calculated for all clocks in the design.
- ❑ `-buffers`—specifies the list of library cells that will be used as buffers in the clock tree
- ❑ `-leaf_max_fanout`—specifies the maximum number of flip-flops that can be driven by a leaf clock buffer
- ❑ `-width` and `-height`—specify the estimated chip dimensions

Note: If you read in a DEF floorplan file for the physical flow, the RC-LP engine can retrieve this information from the design information hierarchy.

The predicted clock tree power consumption takes into account clock-related parameters such as clock period and clock slew specified through the design constraints. The predicted number can be higher before synthesis depending on the number of flip-flops that might be removed during synthesis optimization.

The algorithm assumes a 100 percent toggle rate of the clock.

Example 5-1 on page 87 shows the report that corresponds to the command file shown in Figure 5-2.

Figure 5-2 Command Script for Clock Tree Power Estimation

```
set_attr lib_search_path lib_path
set_attr library {lib1 lib2 ... libn}
set_attribute lef_library {lef1 lef2 ... lefn}
set_attr hdl_search_path hdl_path
read_hdl mydesign.v
elaborate
read_sdc myrc.sdc
read_def my.def.gz
set buf {BUFFD0 BUFFD1 BUFFD2 BUFFD3 BUFFD4 BUFFD6 BUFFD8 BUFFD12 BUFFD16 BUFFD20
BUFFD24 INVD0 INVD1 INVD2 INVD3 INVD4 INVD6 INVD8 INVD12 INVD16 INVD20 INVD24 }
set_attr lp_power_unit mW
report power -clock_tree -buffer $buf -leaf_max_fanout 50
```

Example 5-4 Reporting Clock Tree Power

```
=====
Generated by:           Encounter(r) RTL Compiler 7.1
Generated on:           date
Module:                 tx
Technology libraries:   lib1 200
                        lib2
                        .....
                        libn 1.00
                        physical_cells
Operating conditions:   WCCOM
Interconnect mode:     ple
Area mode:              physical library
=====

Clock Power Estimation Summary
=====

Power of All Clocks
  Dynamic Power          68.173 mW
  Leakage Power          0.000 mW
  Total Power            68.173 mW

Leaf Clock Buffers      357
Total Clock Buffers     759

Estimation Parameters
=====

Clock Buffers Used: BUFFD0 BUFFD1 BUFFD2
                   BUFFD3 BUFFD4 BUFFD6
                   BUFFD8 BUFFD12 BUFFD16
                   BUFFD20 BUFFD24 INVD0
                   INVD1 INVD2 INVD3
                   INVD4 INVD6 INVD8
                   INVD12 INVD16 INVD20
                   INVD24

Max flops driven by one leaf buffer: 50
Die width: 2393.13 um
Die height: 2387.43 um
```

Low Power in Encounter RTL Compiler

Power Analysis

The design used in the previous example has two clocks TCK and cclk. To see the clock tree power for one clock, for example TCK, use the following command:

```
report power -clock_tree TCK -buffer $buf -leaf_max_fanout 50
```

The corresponding report is shown in Example 5-5.

Example 5-5 Reporting Clock Tree Power for Clock TCK

```
=====
Generated by:           Encounter(r) RTL Compiler 7.1
Generated on:           date
Module:                 tx
Technology libraries:   lib1 200
                        lib2
                        .....
                        libn 1.00
                        physical_cells
Operating conditions:   WCCOM
Interconnect mode:     ple
Area mode:              physical library
=====

Clock Power Estimation Summary
=====

Power of Clock 'TCK'
  Dynamic Power          1.733 mW
  Leakage Power          0.000 mW
  Total Power            1.733 mW

Leaf Clock Buffers          9
Total Clock Buffers        19

Estimation Parameters
=====

Clock Buffers Used:  BUFFD0 BUFFD1 BUFFD2
                     BUFFD3 BUFFD4 BUFFD6
                     BUFFD8 BUFFD12 BUFFD16
                     BUFFD20 BUFFD24 INVD0
                     INVD1 INVD2 INVD3
                     INVD4 INVD6 INVD8
                     INVD12 INVD16 INVD20
                     INVD24

Max flops driven by one leaf buffer: 50
Die width: 2393.13 um
Die height: 2387.43 um
```

Reporting on All Power Components

- To report on all power components at once, use the `report power` command:

```
report power [-hier | -flat [-nworst number]] [-depth number] \
  [-sort mode] [instance | net]... [-tcf_summary] [> file]
```

Reporting Power for the Design

The following example reports the power for design m1.

```
rc:/> report power
```

```
=====
...
Module:                m1
Technology library:    slow 1.0
Operating conditions:  slow (balanced_tree)
Wireload mode:        enclosed
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
m1	36	1024.450	965950.648	966975.098
m2	31	918.711	784668.056	785586.767
m3	20	617.363	535027.001	535644.364
m4	17	519.203	438897.419	439416.622
m5	7	205.756	137751.597	137957.353
m5_bbox	0	0.000	33910.310	33910.310
m6	1	12.977	46071.286	46084.263
m3_0	3	98.160	96129.582	96227.742
m3_0_0	3	98.160	96129.582	96227.742
m2myclk	1	3.318	5624.130	5627.448

By default, the report shows the power dissipation for all hierarchical instances.

For each hierarchical *instance* you find:

- The number of cells in that hierarchical instance
- Leakage Power—the total leakage power for the hierarchical instance (in nW).
- Dynamic Power—the sum of the total internal cell power for the hierarchical instance and the net power of all nets dissipated in the hierarchical instance (in nW).
- Total Power—the sum of the leakage power and the dynamic power of the hierarchical instance (in nW).

This report reflects the logical hierarchy of the design. From this report you can tell that the design m1 has 36 cells and one subdesign m2, which has 31 cells. This implies that the design m1 has five leaf cells. Subdesign m2 has two subdesigns: m3 and m2myclk.

Limiting the Number of Hierarchy Levels Shown

If you are only interested in the top-levels of the hierarchy, you can use the `-depth` option to limit the number of levels of hierarchy that you want to report on.

The following example shows 2 levels of hierarchy for `m1`.

```
rc:/> report power -depth 2
```

```
=====
```

```
...
Module:                m1
...
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
m1	36	1024.450	965950.648	966975.098
m2	31	918.711	784668.056	785586.767
m3	20	617.363	535027.001	535644.364
m2myclk	1	3.318	5624.130	5627.448

Reporting the Power of the Leaf Instances

To get a more detailed report and see how the power is distributed over the leaf instances of the reported hierarchical instances, you can use the `-flat` option.

```
rc:/designs/m1> report power -depth 2 -flat
```

```
=====
```

```
...
Technology library:    slow 1.0
...
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
m2/o_m2_clk2_1_reg_0		34.558	30136.644	30171.202
m2/o_m2_clk2_1_reg_1		34.297	30136.554	30170.851
o_m1_clk1_0_reg_2		33.703	30107.017	30140.720
m2/o_m2_clk1_0_reg_0		33.428	32131.758	32165.186
o_m1_clk1_0_reg_1		33.270	30136.554	30169.824
m2/o_m2_clk1_0_reg_1		32.887	32220.370	32253.257
m2/o_m2_clk1_0_reg_2		32.814	32043.146	32075.960
o_m1_clk1_0_reg_0		32.605	30166.047	30198.651
m2/o_m2_clk2_0_reg_1		32.441	15508.875	15541.315
m2/o_m2_clk2_1_reg_2		32.430	30136.554	30168.984
m2/o_m2_clk2_0_reg_0		32.318	15508.875	15541.193
m2/o_m2_clk2_0_reg_2		29.776	15561.573	15591.349
m2/m2myclk/g8		3.318	5624.130	5627.448
m2/g15		3.081	10632.578	10635.659
g6		3.081	12851.198	12854.280
g10		3.081	10632.578	10635.659

Low Power in Encounter RTL Compiler

Power Analysis

Because both the `-flat` option and the `-depth` option were specified, the report only shows the power dissipated in the five leaf cells of m1 and in the leaf instances of m2 up to two levels deep. By looking at the report in [Limiting the Number of Hierarchy Levels Shown](#) on page 93, you can determine that m2 has 10 leaf instances.

Reporting Power for Specific Instances

If you are interested in the power of a particular instance, you can specify the RC path name or the Verilog path to that instance. You need to specify the path starting from the current location in the design hierarchy from where the report command is issued.

The following example shows you how to request the power report for the hierarchical instance m4. The current location in the hierarchy is `/designs/m1`. The RC path can contain wildcards, as long as the path is uniquely defined.

```
rc:/designs/m1> report power inst*_hier/m2/i*_hier/m3/ins*_hier/m4
```

or

```
rc:/designs/m1> report power m2/m3/m4
```

```
=====
...
Module:                m1
...
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
m4	17	519.203	438897.419	439416.622
m5	7	205.756	137751.597	137957.353
m5_bbox	0	0.000	33910.310	33910.310
m6	1	12.977	46071.286	46084.263

Sorting According to One Power Component

If you want to sort the report according to one specific power component, you can use the `-sort` option. The following modes are available for instance-based reports:

<code>internal</code>	Sorts by descending internal power.
<code>leakage (default)</code>	Sorts by descending leakage power.
<code>net</code>	Sorts by descending net power.
<code>switching</code>	Sorts by descending total switching power, which is the sum of the internal and net power.

Low Power in Encounter RTL Compiler

Power Analysis

The following example sorts the report according to descending internal power:

```
rc:/designs/ml> report power -flat -sort internal
```

```
=====
```

```
...
```

```
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)

m2/m3/m4/o..._clk3_0_reg_1		33.302	32259.638	32292.940
m2/o_m2_clk1_0_reg_1		32.887	32220.370	32253.257
m2/m3/m4/o..._clk1_0_reg_0		33.490	32190.832	32224.322
m2/m3/m4/o..._clk3_0_reg_0		33.188	32141.607	32174.796
m2/m3/m4/o..._clk2_1_reg_1		33.681	32131.758	32165.439
m2/o_m2_clk1_0_reg_0		33.428	32131.758	32165.186
m2/m3/m4/o..._clk2_1_reg_0		33.641	32072.683	32106.325
m2/m3/m3_0..._clk1_0_reg_1		32.397	32072.683	32105.080
m2/m3/m3_0..._clk1_0_reg_2		33.005	32043.242	32076.247
m2/m3/m4/o..._clk1_0_reg_2		33.839	32043.146	32076.985
m2/o_m2_clk1_0_reg_2		32.814	32043.146	32075.960
m2/m3/m4/o..._clk3_0_reg_2		33.420	32023.388	32056.808
m2/m3/m4/o..._clk1_0_reg_1		32.157	32013.657	32045.814
m2/m3/m4/o..._clk2_1_reg_2		33.752	32013.657	32047.409
m2/m3/m3_0..._clk1_0_reg_0		32.758	32013.657	32046.415
...				
m2/m3/m4/m5/o_m5_1_reg_1		34.194	1058.807	1093.001

The example shows that some of the instance paths are not completely shown because they are too long. To show the full path you can specify the `-full_instance_names` option.

```
rc:/designs/ml> report power -flat -sort internal -full_instance_names
```

```
=====
```

```
...
```

```
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)

m2/m3/m4/o_m4_clk3_0_reg_1		33.302	32259.638	32292.940
m2/o_m2_clk1_0_reg_1		32.887	32220.370	32253.257
m2/m3/m4/o_m4_clk1_0_reg_0		33.490	32190.832	32224.322
m2/m3/m4/o_m4_clk3_0_reg_0		33.188	32141.607	32174.796
m2/m3/m4/o_m4_clk2_1_reg_1		33.681	32131.758	32165.439
m2/o_m2_clk1_0_reg_0		33.428	32131.758	32165.186
m2/m3/m4/o_m4_clk2_1_reg_0		33.641	32072.683	32106.325
m2/m3/m3_0/m3_0_0/o_m3_clk1_0_reg_1		32.397	32072.683	32105.080
m2/m3/m3_0/m3_0_0/o_m3_clk1_0_reg_2		33.005	32043.242	32076.247
m2/m3/m4/o_m4_clk1_0_reg_2		33.839	32043.146	32076.985
m2/o_m2_clk1_0_reg_2		32.814	32043.146	32075.960
m2/m3/m4/o_m4_clk3_0_reg_2		33.420	32023.388	32056.808
m2/m3/m4/o_m4_clk1_0_reg_1		32.157	32013.657	32045.814
m2/m3/m4/o_m4_clk2_1_reg_2		33.752	32013.657	32047.409
m2/m3/m3_0/m3_0_0/o_m3_clk1_0_reg_0		32.758	32013.657	32046.415
...				
m2/m3/m4/m5/o_m5_1_reg_1		34.194	1058.807	1093.001

Low Power in Encounter RTL Compiler

Power Analysis

Net-based Power Reports

You can also request net-based power reports.

The following example requests a power report for all top-level enable nets. A report for 10 enable nets is given.

```
rc:/designs/ml> report power [find . -max 2 -net en*]
=====
...
Technology library:      slow 1.0
Operating conditions:    slow (balanced_tree)
Wireload mode:          enclosed
=====

      Net
Net    Power (nW) Prob. Rate (/ns) Cap. (nF)
-----
en_1[0]    458.395 0.500    0.020    39.000
en_1[1]    458.395 0.500    0.020    39.000
en_1[2]    458.395 0.500    0.020    39.000
en_1[3]    458.395 0.500    0.020    39.000
en_1[4]    458.395 0.500    0.020    39.000
en_1[5]    458.395 0.500    0.020    39.000
en_1[6]    458.395 0.500    0.020    39.000
en_1[7]    458.395 0.500    0.020    39.000
en_1[8]    458.395 0.500    0.020    39.000
en_1[9]    458.395 0.900    0.020    39.000
```

If the results would be quite different, you can also sort net-based power reports by using the `-sort` option. The following modes are available for net-based reports:

<code>load</code>	Sorts by descending capacitive load on the net.
<code>prob</code>	Sorts by descending probability of the nets being high.
<code>rate</code>	Sorts by descending toggle rates of the nets.
<code>switching</code> (default)	Sorts by descending net power.

Showing User-Asserted Information

The power report can show an additional summary which includes the number of nets with user-asserted switching activities. If you request a net-based power report, each user-asserted net is marked with an asterisk (*).

Reporting Power on Leaf Instances

- To report power on some specific leaf instances, use the `-power` option of the `report instance` command:

```
report instance [-timing] [-power] instance... [>file]
```

The following example requests a power report for leaf instance `o_m2_clk2_1_reg_0`.

```
rc:/> report instance -power [find / -inst o_m2_clk2_1_reg_0]
```

```
=====
Generated by:      RTL Compiler-D (RC) version
Generated on:      date
Module:            m1
Technology library: slow 1.0
Operating conditions: slow (balanced_tree)
Wireload mode:     enclosed
=====
```

Instance Power Info

Instance m2/o_m2_clk2_1_reg_0 of Libcell EDFFX1

Leakage Power(nW)	Internal Power(nW)	Net Power(nW)
34.558	30020.004	116.640

Pin	Net	Computed Probability	Computed Toggle Rate(/ns)	Net Power(nW)
Q	o_m2_clk2_1[0]	0.8	0.0200	116.640
CK	n_0	0.5	1.8725	8845.540
D	in_2[21]	0.5	0.0200	136.469
E	en_2[7]	0.5	0.0200	458.395

Arc From	Arc To	When	Arc Activity	Arc Power(nW)
CK	CK	!D & !E	0.4659	13657.500
CK	CK	D & !E	0.4704	13585.150
CK	CK	!D & E	0.4659	17619.850
CK	CK	D & E	0.4704	17732.100
D	D		0.0200	14343.100
E	E		0.0200	15564.250
CK	Q		0.0200	5982.895

Note: In case the switching activities are user-asserted, the values are appended with an asterisk (*).

Reporting Power Consumption per Used Cell Types

- To report the technology library cells that were implemented (and their originating libraries) and the power consumed by cell type, use the `-power` option of the `report gates` command:

```
report gates [-power] [design] [> file]
```

The following example shows the power information by cell type. The first table lists the leakage and internal power of all instances per used library cell. The second table shows the leakage and internal power of all sequential cells, inverters, and combinational cells, as well as the percentage of power that each type consumes.

```
rc:/> report gates -power
```

```
=====
...
Module:                ml
...
=====
```

Gate	Instances	Area	Leakage Power (nW)	Internal Power (nW)	Library
ADDFX1	7	249.518	0.007	12229.479	typical
...					
XOR2X1	3	35.645	0.003	1011.269	typical
total	121	1705.887	0.072	89593.045	

Type	Instances	Area	Area %	Leakage Power (nW)	Leakage Power %	Internal Power (nW)	Internal Power %
timing_model	2	47.527	2.8	0.002	2.8	7501.062	8.4
sequential	16	427.745	25.1	0.016	22.2	39047.131	43.6
inverter	11	37.343	2.2	0.000	0.0	1195.550	1.3
buffer	3	20.369	1.2	0.003	4.2	512.756	0.6
logic	89	1172.903	68.8	0.051	70.8	41336.546	46.1
total	121	1705.887	100.0	0.072	100.0	89593.045	100.0

If your design uses different technology libraries, a third table will show the number and percentage of instances coming from each library.

If you insert clock-gating logic in the design, the number of sequential instances used in the design may or may not increase depending on how the clock-gating logic is selected:

- If you use an integrated clock-gating cell from the technology library, this cell is treated as a `timing_model` (as shown in the example above). In this case the number of sequential instances before and after clock-gating insertion does not change.
- If the requested integrated clock-gating cell does not exist in the technology library, the RC-LP engine can create discrete clock-gating logic. In this case the latch or flip-flop used for clock-gating will be accounted for in the report.

Reporting on Specific Power Components for Specific Objects

The RC-LP engine also provides several analysis attributes to allow you to get specific information on specific objects.

Important

Some info and warning messages, which are displayed when calculating the power of an object, are not always redisplayed when recalculating the power. In RTL Compiler, power is computed on demand and cached (stored). Therefore, if the netlist is not changed, the value is not recomputed and the stored value will be displayed.

This section uses the same example as used in [Reporting on All Power Components](#).

Getting Power-Related Information on the Design

- To get the internal power on the design, use the following command:

```
rc:/designs/ml> get_attr lp_internal_power  
819893.893949
```

- To get the leakage power on the design, use the following command:

```
rc:/designs> get_attr lp_leakage_power ml  
1024.449807
```

- To get the net power on the design, use the following command:

```
rc:/> get_attr lp_net_power ml  
146056.753800
```

- To find out the default toggle rate specified on the design, use the following command:

```
rc:/> get_attr lp_default_toggle_rate ml  
0.020000
```

- To find out the default probability specified on the design, use the following command:

```
rc:/> get_attr lp_default_probability ml  
0.50000
```

All these attributes are design attributes, so you need to make sure that you give the path to the design dependent on your current location in the hierarchy.

Getting Power-Related Information on an Instance

- To get the internal power on instance m2, use the following commands:

```
rc:/> cd /designs/ml
rc:/designs/ml> get_attr lp_internal_power m2
726116.379831
```

- To get the leakage power on instance m2, use the following command:

```
rc:/designs> get_attr lp_leakage_power m1/m2
918.710523
```

- To get the net power on instance g5, use the following command:

```
rc:/> get_attr lp_net_power [find / -maxdepth 10 -instance g6]
10920.420000
```

You can find out what the default switching activities are for a hierarchical instance.

- To find out the default toggle rate specified on the hierarchical instance, use the following command:

```
rc:/designs/ml> get_attr lp_default_toggle_rate */instances_hier/m2
0.020000
```

- To find out the default probability specified on the subdesign, use the following command:

```
rc:/designs/ml> get_attr lp_default_probability */instances_hier/m2
0.50000
```

All these attributes are instance attributes, so you need to make sure that you give the path to the instance dependent on your current location in the hierarchy.

Getting Power-Related Information on a Pin, Port, or Subport

The only information you can get on a pin, port, or subport is the switching power dissipated on the net connected to this pin, port, or subport.

- To get the power dissipated in the net connected to port clk3, use the following command:

```
rc:/designs/ml> get_attr lp_net_power [find . -max 3 -port clk3]
11897.280000
```

Getting Power-Related Information on a Net

- To get the net power on net clk3, use the following command:

```
rc:/> get_attr lp_net_power [find . -max 3 -net clk3]
11897.280000
```

Low Power in Encounter RTL Compiler

Power Analysis

- To get the net power on net `en_1[0]`, use the following command:

```
rc:/> get_attr lp_net_power /designs/m1/nets/en_1[0]
458.395200
```

- To find out the toggle rate on net `en_1[0]`, use the following command:

```
rc:/> get_attr lp_computed_toggle_rate [find . -max 5 -net en_1[0]]
0.020000
```

- To find out the source of the toggle rate on net `en_1[0]`, use the following command:

```
rc:/> get_attr lp_toggle_rate_type [find . -max 5 -net en_1[0]]
default
```

- To find out the probability on net `clk3`, use the following command:

```
rc:/> get_attr lp_computed_probability [find . -max 5 -net clk3]
0.50000
```

- To find out the source for the probability on net `clk3`, use the following command:

```
rc:/> get_attr lp_probability_type [find . -max 5 -net clk3]
clock
```

Example 5-6 Probability and Toggle Rate of Net Driven by a Clock Port

If the net is driven by a clock port, the probability and toggle rate can be derived from the duty cycle of the clock.

```
rc:/> define_clock -period 3500 -name clock -rise 0 -fall 75 -design /*/alu
...
rc:/> get_attr lp_computed_probability */nets/clock
0.75000
rc:/> get_attr lp_computed_toggle_rate */nets/clock
0.571429
```

The clock is defined as high for 75 percent of the clock period (`-fall` value divided by `-divide_fall` value is 75/100), the probability of the clock being high is 0.75.

Because the default toggle rate unit is /ns and the period is expressed in ps, the toggle rate can be determined by the reverse of the clock period times 2 (because the signal toggles two times per period) or 2/3.5 ns.

Troubleshooting

Common Causes of Correlation Problems

Debugging Techniques To Find Cause of Correlation Problem

Changing Clock Frequency Does not Have Expected Result

Common Causes of Correlation Problems

Correlation Between Two Sessions of RTL Compiler

You can get different power results when reporting the power on the same netlist in two different sessions.

To report matching power results, ensure that you use the same

- Design and timing constraints in both sessions
- Switching activities in both sessions

To use the same switching activities in both sessions:

- a. Write out a TCF file containing the computed switching activities in the first session:

```
write_tcf -computed > computed.tcf
```

- b. Read in this TCF file in the second session:

```
read_tcf computed.tcf
```

Correlation Between RTL Compiler and Other Power Analysis Tools

You might see some differences between the power estimation results of RTL Compiler and other tools. This section describes some possible causes.

Tools Read Different Switching Activities

Make sure that both power analysis tools read in the same switching activities.

Tools Use Different Default Values For Switching Activities

Check the default values for the switching activities in both power analysis tools and run power analysis again with the same settings.

- The default probability in RTL Compiler (set through the `lp_default_probability` attribute) is 0.5.
- The default toggle rate in RTL Compiler (set through the `lp_default_toggle_rate` attribute) is 0.02 per nanosecond.

Tools Treat Clock Nets and Asynchronous Set and Reset Nets of Sequential Gates Differently

By default, RTL Compiler treats clock nets and asynchronous set and reset nets of sequential gates as ideal. This has the following implications:

- The slew rate on the corresponding pins will be 0.
This can cause differences in the *internal* power results if the other tool uses non-zero slew values on those pins.
- The net capacitance of these nets will be zero.
This can cause differences in the *net* power results if the other tool does not treat the net as ideal.

Tools Use Different Wireload Model

The net power and internal power results depend on the wire capacitance. If the power analysis tools you are comparing use different wireload models, these results will be different.

Tools Use Different Switching Activities for Clock Nets

If no user-asserted switching activities are specified for a net driven by a clock port, RTL Compiler derives the probability and toggle rate from the duty cycle of the clock, while other tools might use the default values for these nets.

Debugging Techniques To Find Cause of Correlation Problem

If you checked all possible causes listed in section [Common Causes of Correlation Problems](#), and the results still do not correlate, you can try the following debugging techniques:

1. Find the instances that consume the most power:

```
report power -nworst 10 -sort internal [find / -inst instances_seq/*]
```

2. Report the power information for one instance:

```
report instance -power instance
```

Check the information in this report.

Changing Clock Frequency Does not Have Expected Result

Changing the clock frequency to half of the value does not necessarily reduce the switching power for the whole design to half the value.

Changing the clock frequency only affects the power estimation on the clock net. Power estimation on the whole design is also based on propagation of the switching activities on the primary input pins.

Clock Gating

- [Introduction](#) on page 106
- [RTL Coding Style Examples](#) on page 109
- Tasks
 - [Enabling Clock Gating](#) on page 114
 - [Controlling Handling of Timing Exceptions during Clock Gating](#) on page 114
 - [Controlling Naming of Modules, Nets, and Ports](#) on page 114
 - [Controlling Selection of Clock-Gating Logic](#) on page 115
 - [Controlling Insertion of Clock-Gating Logic](#) on page 130
 - [Previewing the Effect of Clock Gating](#) on page 133
 - [Clock Gating with DFT](#) on page 135
 - [Checking System Messages during Optimization](#) on page 144
 - [Reporting Clock-Gating Information](#) on page 146
 - [Clock-Gating Post-Processing](#) on page 152
 - [Multi-Stage Clock Gating](#) on page 159
 - [Inserting Clock Gating in a Mapped Netlist](#) on page 170
- [Troubleshooting](#) on page 173
- [Limitations](#) on page 174

Introduction

Even though data is loaded into registers very infrequently in most designs, the clock signal continues to toggle at every clock cycle. Often, the clock signal also drives a large capacitive load, making clock signals a major source of dynamic power dissipation. Gating a group of flip-flops that are enabled by the same control signal reduces unnecessary clock toggles.

Clock gating reduces power dissipation for the following reasons:

- Power is not dissipated during the idle period when the register is shut off by the gating function.
- Power is saved in the gated-clock circuitry.
- The logic on the enable circuitry in the original design is removed.

Figure 6-1 shows how clock gating can help improve power dissipation of a circuit. The clock-gating logic along with the set of registers that it is controlling is referred to as the clock-gating domain.

Figure 6-1 Clock Gating Concept

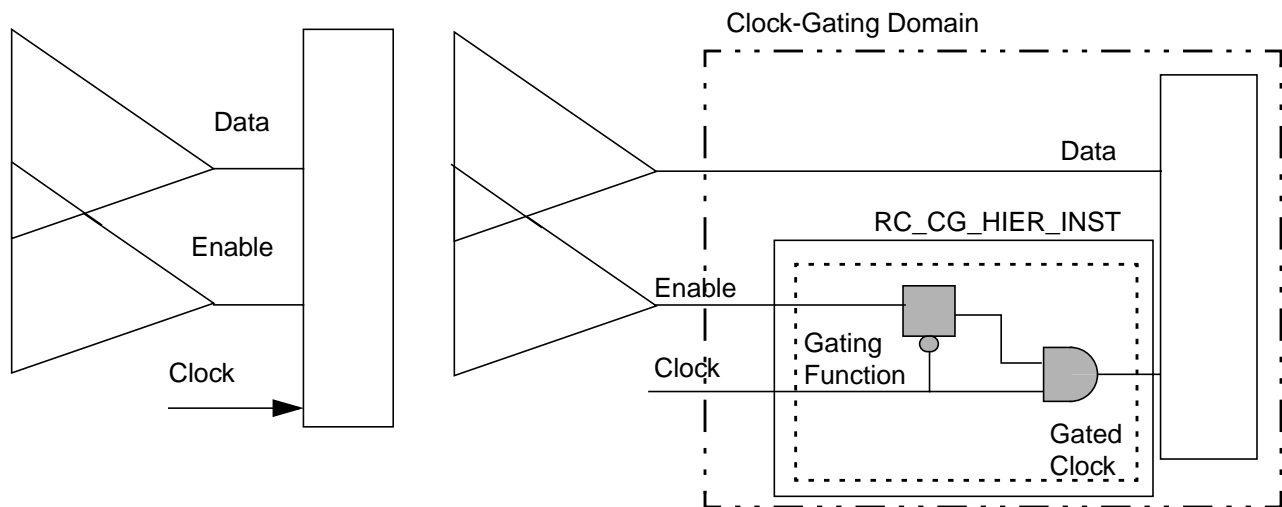
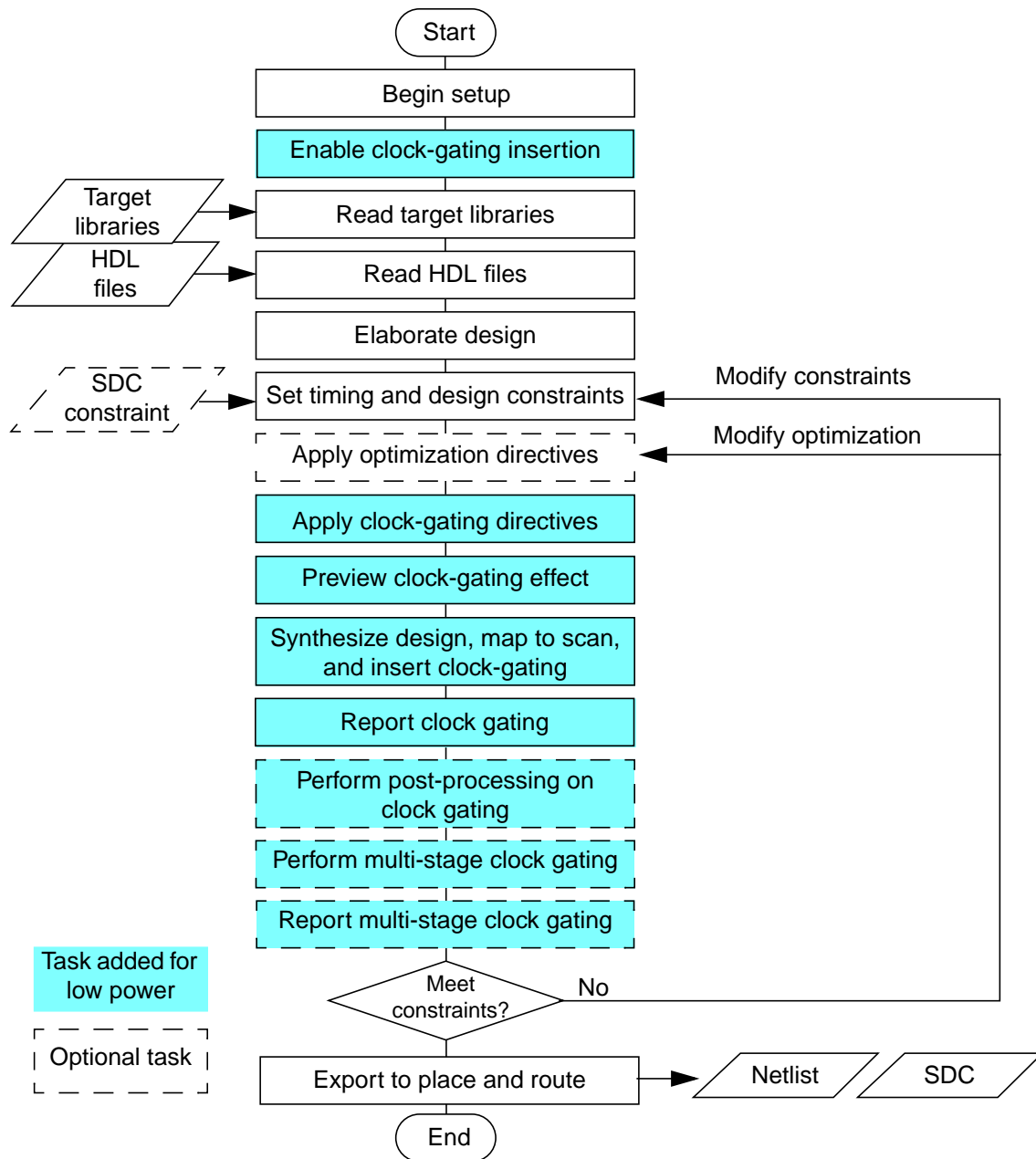


Figure 6-2 on page 107 highlights the tasks you need to add to the generic synthesis flow to use the clock-gating flow starting from RTL.

Low Power in Encounter RTL Compiler

Clock Gating

Figure 6-2 Recommended Clock-Gating Flow Starting from RTL



Tasks for the Recommended Clock-Gating Flow Starting from RTL

■ Enabling Clock Gating before elaboration

During elaboration, given the RTL netlist of the design, the RC-LP engine identifies the registers for clock gating and determines when those registers are inactive.

Low Power in Encounter RTL Compiler

Clock Gating

- Applying the clock-gating directives including
 - ❑ [Controlling Handling of Timing Exceptions during Clock Gating](#) on page 114
 - ❑ [Controlling Selection of Clock-Gating Logic](#)
 - ❑ [Controlling Insertion of Clock-Gating Logic](#)

If clock gating is enabled the RC-LP engine inserts one gating logic per register bank with the same enable signal for clocks in the inactive periods before mapping.



Tip

Because several enable muxes can be replaced with one clock-gating logic, clock-gating insertion can also result in a reduction of area.

- [Previewing the Effect of Clock Gating](#) after applying the clock-gating directives

You can check the potential reduction in the clock toggling when clock-gating logic would be inserted—without making any modifications to the netlist.
- [Reporting Clock-Gating Information](#) after synthesis
- [Clock-Gating Post-Processing](#) such as decloning or removing clock gating.
- [Multi-Stage Clock Gating](#) such as creating shared clock-gating logic and splitting clock-gating instances into multiple levels of clock-gating logic.

Other Flows

If you want to use clock gating with DFT, refer to [Clock Gating with DFT](#) for more information.

If you want to insert clock gating in a mapped netlist, refer to [Inserting Clock Gating in a Mapped Netlist](#) for more information.

If you want to use clock gating with retiming, refer to [Retiming the Design](#) in *Using Encounter RTL Compiler*.

RTL Coding Style Examples

If you do not infer the synchronous set or reset signal on the flip-flop, the RC-LP engine will not use the synchronous set or reset in the clock-gating logic (see [Example 6-1](#) on page 110).

If the flip-flops were inferred with a synchronous reset as shown in [Example 6-2](#) on page 111 and [Figure 6-5](#) on page 111, the RC-LP engine will combine the reset signal with the synchronous enable signal to create a new signal as shown in [Figure 6-6](#) on page 111. The RC-LP engine then uses the new signal as the enable for the clock-gating logic.

If you infer an asynchronous set, the RC-LP engine will not use the asynchronous set in the clock-gating logic (see [Example 6-3](#) on page 112).

For more information on the required pragmas or synthesis directives, refer to [Set and Reset Synthesis Pragmas](#) in *HDL Modeling in Encounter RTL Compiler*.

[Example 6-4](#) on page 113 illustrates that even when the enable logic resides outside the logic hierarchy, clock gating can be inserted. In this case, the RC-LP engine creates an additional port during clock gating and combines the enable signal with the clock signal to create the clock gating logic in the hierarchy where the registers reside.

Low Power in Encounter RTL Compiler

Clock Gating

Example 6-1 RTL of Flip-Flops *without* Inferred Synchronous Reset

```
module top(en, in1, in2, clk, out1)
input en, clk;
input [7:0] in1, in2
output [7:0] out1;
reg [7:0] out1;
always @ (posedge clk)
    if (en)
        out1 <= in1 + in2;
endmodule
```

Figure 6-3 Circuit with Flip-Flops *without* Inferred Synchronous Reset

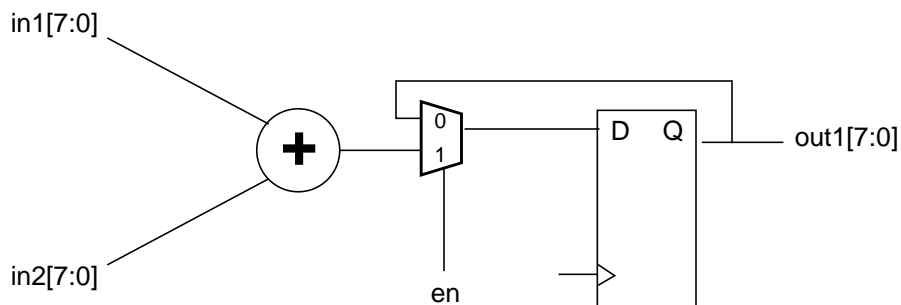
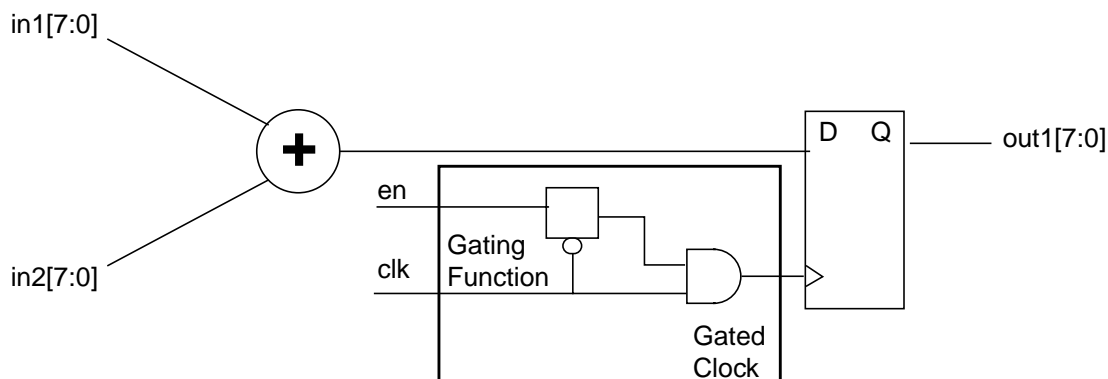


Figure 6-4 Circuit with Flip-Flops *without* Inferred Synchronous Reset with Clock Gating



Example 6-2 RTL of Flip-Flops Inferred with a Synchronous Reset

```
module top(en, rst, in1, in2, clk, out1)
input en, rst, clk;
input [7:0] in1, in2;
output [7:0] out1;
reg [7:0] out1;
//cadence synchro_reset "rst"
always @ (posedge clk)
    if (rst)
        out1 <= 0;
    else
        if (en)
            out1 <= in1 + in2;
endmodule
```

Figure 6-5 Circuit with Flip-Flops Inferred with a Synchronous Reset

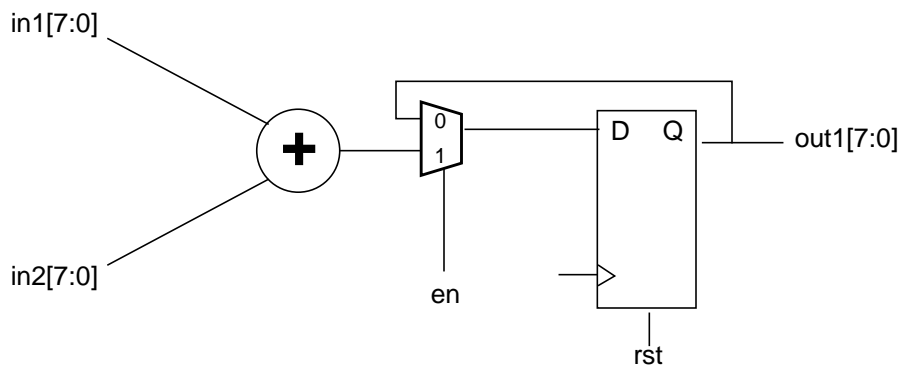
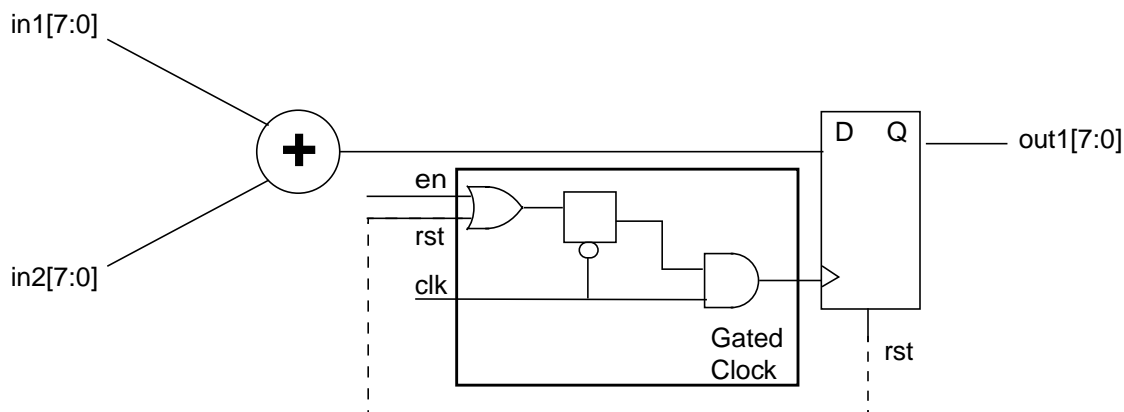


Figure 6-6 Circuit with Flip-Flops Inferred with a Synchronous Reset with Clock Gating



Example 6-3 RTL of Flip-Flops Inferred with an Asynchronous Reset

```
module top(en, a_rst, in1, in2, clk, out1)
input en, a_rst, clk;
input [7:0] in1, in2;
output [7:0] out1;
reg [7:0] out1;

always @ (posedge clk or posedge a_rst)
    if (a_rst)
        out1 <= 0;
    else
        if (en)
            out1 = in1 + in2;
endmodule
```

Figure 6-7 Circuit with Flip-Flops Inferred with an Asynchronous Reset

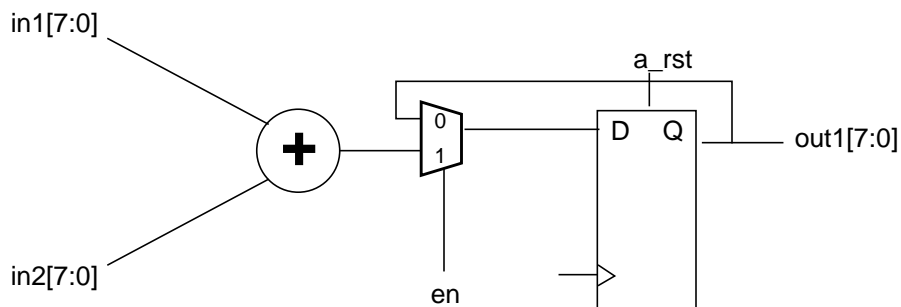
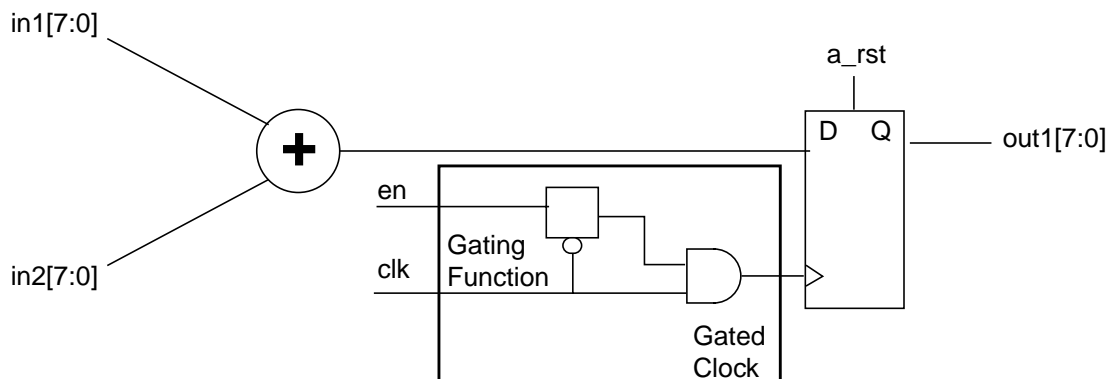


Figure 6-8 Circuit with Flip-Flops Inferred with an Asynchronous Reset with Clock Gating



Low Power in Encounter RTL Compiler

Clock Gating

Example 6-4 Example of Hierarchical Clock Gating

```
module test (a,b,dt,clk) ;
input a;
output dt;
input b;
input clk;
reg dt;

    wire d_in = (b) ? a : dt ;
    my_ff my_ff(.q(dt), .d(d_in), .clk(clk));

endmodule

module my_ff (q, d, clk);
input clk;
input d;
output q;
reg q;

    always @(posedge clk) begin
        q<=d;
    end
endmodule
```

Figure 6-9 Hierarchical Clock Gating Example before Clock Gating

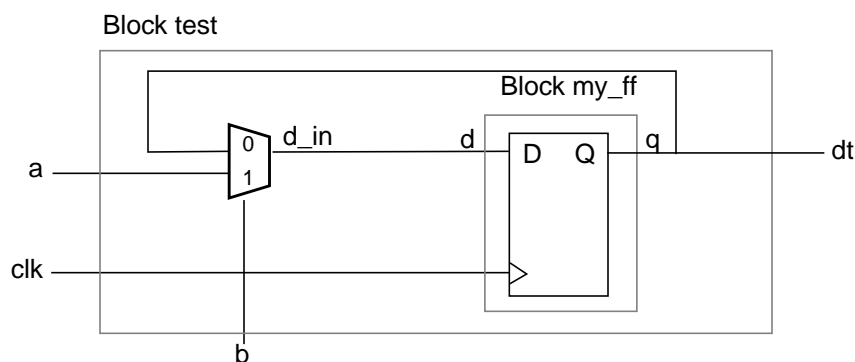
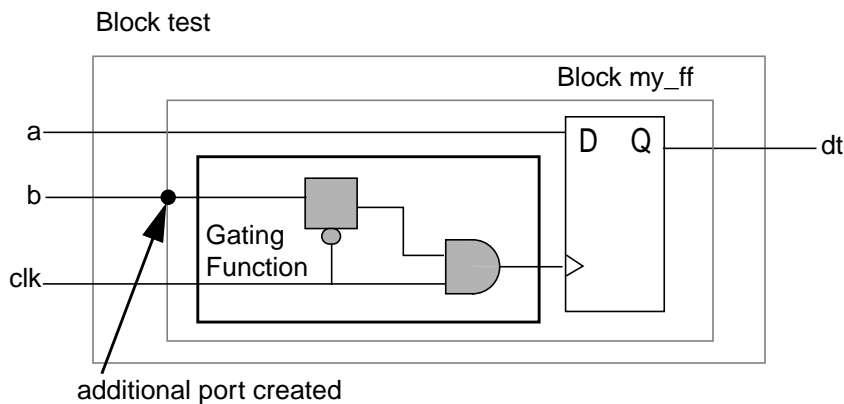


Figure 6-10 Hierarchical Clock Gating Example after Clock Gating



Enabling Clock Gating

To ensure that clock-gating logic is inserted during synthesis, set the following attribute:

```
set_attribute lp_insert_clock_gating true /
```



Tip

Set this attribute before the `elaborate` command, to achieve the maximum percentage of gated flip-flops.

Controlling Handling of Timing Exceptions during Clock Gating

To ensure that timing exceptions set on flop instances or on their clock, enable, and reset pins are taken into account during clock gating, set the following attribute before you synthesize the design:

```
set_attribute lp_clock_gating_exceptions_aware true /
```

Setting this attribute to `false` indicates that any flops driven by the same clock, enable and reset signals can be gated by the same clock-gating instance.

If you set this attribute to `true`, any flops that have different exceptions (on the instance or pins) will be gated by different clock-gating instances even if they are driven by the same clock, enable and reset signals. The timing exceptions are copied to the clock-gating instance or its pins.

See also [Excluded from clock gating](#) for more information.

Note: Depending on the settings of the `lp_clock_gating_min_flops` and `lp_clock_gating_max_flops` attributes, setting the `lp_clock_gating_exceptions_aware` attribute to `true` might increase or decrease the number of clock-gating instances.

Controlling Naming of Modules, Nets, and Ports

To specify the prefix to be added to all clock-gating modules, generated clock nets, and the ports created by clock-gating insertion, use the `lp_clock_gating_prefix` attribute. By default, no prefix is added.

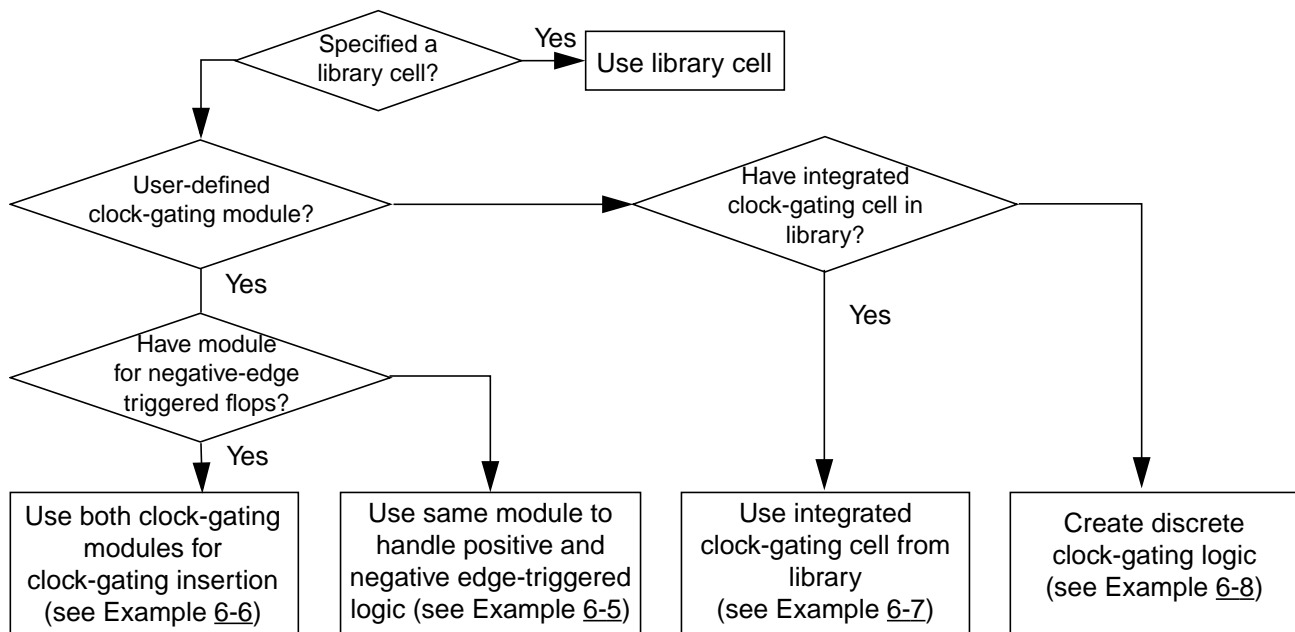
Controlling Selection of Clock-Gating Logic

You can choose clock-gating logic by

- Specifying the Library Cell Name of the Integrated Clock-Gating Cell
- Defining a Clock-Gating Module
- Using Attributes to Select an Integrated Clock-Gating Cell in the Technology Library

Figure 6-11 shows the process that the RC-LP engine follows to select the logic. As illustrated in the flowchart, the RC-LP engine can create discrete clock-gating logic if the requested integrated clock-gating cell does not exist in the technology library.

Figure 6-11 Selection of Clock-Gating Logic



If you use a user-defined clock-gating module, you find the clock-gating module in the design hierarchy at:

`/designs/module`

If you do not use a user-defined clock-gating module, the RC-LP engine creates a separate subdesign for the clock-gating logic in each clock-gating domain. You can find the clock-gating logic subdesigns in the design hierarchy at

`/designs/design/subdesigns/RC_CG_MODx`

Note: In case of discrete clock-gating logic, the created clock-gating modules are called `RC_CG_MOD_AUTO_design` and `RC_CG_MOD_AUTO_design_neg`.

Low Power in Encounter RTL Compiler

Clock Gating

Table 6-1 shows the fixed names that the RC-LP engine uses or expects you to use for the port names of the clock-gating module. The number of ports of the clock-gating module depends on the contents of the clock-gating logic.

Table 6-1 Clock-Gating Module Ports

Name	Definition	Comment	Related Attribute
a_rst	asynchronous reset port	Exists only if the gated flip-flops have an asynchronous set/reset pin and reset logic was added to the glitch preventing logic	<u>lp_clock_gating add reset</u>
ck_in	clock input port	Standard port	
ck_out	clock output port	Standard port	
enable	synchronous enable port	Standard port	
obs	observation port	Exists only if the gating logic also contains observability logic	<u>lp_clock_gating add obs port</u>
s_rst	synchronous reset port	Exists only if the gated flip-flops have a synchronous set or reset signal.	
test	test port	Exists only if the gating logic also contains test logic	<u>lp_clock_gating control point</u>

You can find the instance of each clock-gating logic subdesign in the design hierarchy at

`/designs/design/instances_hier/*/RC_CG_HIER_INSTx`

For example, to find all clock-gating instances in the hierarchy, you can use the following command:

```
rc:/> find . -inst RC_CG_HIER_INST*  
/designs/top/instances_hier/u_a/instances_hier/RC_CG_HIER_INST /designs/top/  
instances_hier/u_b/instances_hier/RC_CG_HIER_INST
```

A gated-clock net can be identified through the `rc_gclk` string in its name.

Important

Because the RC-LP engine uses the clock-gating hierarchy as a vehicle to recognize the clock-gating logic inserted by RTL Compiler, you should not use the `ungroup` command, as it removes the clock-gating hierarchy.

Specifying the Library Cell Name of the Integrated Clock-Gating Cell

You can specify which library cell you want to use for clock gating.

- To identify this cell, set the following attribute on the design or on a list of subdesigns:

```
set_attribute lp_clock_gating_cell path_name_for_cell  
{design|subdesign_list}
```

The path name is the path to this cell in the RTL Compiler design hierarchy. For example,

```
set_attribute lp_clock_gating_cell [find / -libcell cgic-cell] /designs/design  
set_attribute lp_clock_gating_cell /designs/libraries/yyy/libce*/cgic1 \  
[find / -subdesign bot]
```

The attribute set at a subdesign takes precedence over the attribute set at the top design.

To prevent that the specified cell is resized and replaced with another cell during optimization, the RC-LP engine sets the value of the `preserve` attribute to `delete_ok` on the corresponding clock-gating instance.

Note: Only specify a library cell that has the `clock_gating_integrated_cell` attribute in the library (`.lib`). Make sure that both the `dont_use` and `dont_touch` cell attributes are set to `false` in the library.

If the specified cell does not exist in any of the libraries, the auto clock-gating insertion will fail. If multiple cells are found, the tool reports an error.



This attribute overrides the attributes that select an integrated clock-gating cell from the technology library (see [Using Attributes to Select an Integrated Clock-Gating Cell in the Technology Library](#)).

Defining a Clock-Gating Module

You can create a separate module to define your own customized clock-gating logic and request the tool to instantiate it as the clock-gating logic. The module description can be a description of the discrete elements in RTL, or a generic or mapped netlist.

If the module contains mapped gates, the RC-LP engine sets the value of the `preserve` attribute to `size_delete_ok` for this module.

- To specify the name of the clock-gating module, set the following attribute on the design or on a list of subdesigns:

```
set_attribute lp_clock_gating_module clock_gating_module_name  
{design|subdesign_list}
```

Low Power in Encounter RTL Compiler

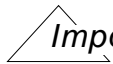
Clock Gating

The attribute set at a subdesign takes precedence over the attribute set at the top design.



Tip

If you have both the `lp_clock_gating_cell` and `lp_clock_gating_module` attributes set on the design or the same subdesign, the `lp_clock_gating_cell` attribute takes precedence.



Important

When RTL Compiler uses a user-defined clock-gating module, RTL Compiler does not check the logic inside this module. As a result, you might get a non equivalent report when performing logic equivalence checking with the original RTL.

Requirements for a user-defined clock gating module

- Make sure that the logic inside the specified module is correct, because the RC-LP engine does not check the correctness of the logic defined in the module.
- When setting the `clock_gating_module` attribute, always specify the name of the module for the clock-gating logic for the positive-edge triggered registers. [Figure 6-15](#) on page 121 shows where to specify this attribute in the flow.
- [Example 6-5](#) on page 119 shows how the RC-LP engine handles negative-edge triggered registers, if you have only one clock-gating module (for the positive-edge triggered registers). [Example 6-6](#) on page 120 shows an example of clock gating using clock-gating modules for positive-edge and negative-edge triggered registers.
- If you also created clock-gating logic for negative-edge triggered registers, you must use the following naming convention. If you use `module` for the clock-gating logic for the positive-edge triggered registers, use `module_neg` for the clock-gating logic for negative-edge triggered registers.
- The module *must* contain the following ports: clock input port, clock output port, and enable port. Use the fixed port names for the clock-gating module. Refer to [Table 6-1](#) on page 116 for a complete list of the fixed port names.



Important

If used, the `a_rst`, `enable`, `s_rst` and `test` pins must be active high.

- If the flip-flops to be gated were inferred with a synchronous set or reset, the clock-gating module must also contain a synchronous set or reset port. The RC-LP engine will connect the synchronous reset ports.

Low Power in Encounter RTL Compiler

Clock Gating

As mentioned in [RTL Coding Style Examples](#), if the flip-flops were inferred with a synchronous reset, the RC-LP engine inserts a gate to combine the reset signal with the synchronous enable signal and uses the new signal as the enable for the clock-gating logic.

Note: However, if the user-defined clock-gating module is *mapped*, the RC-LP engine assumes that the module contains the necessary logic already.

Important

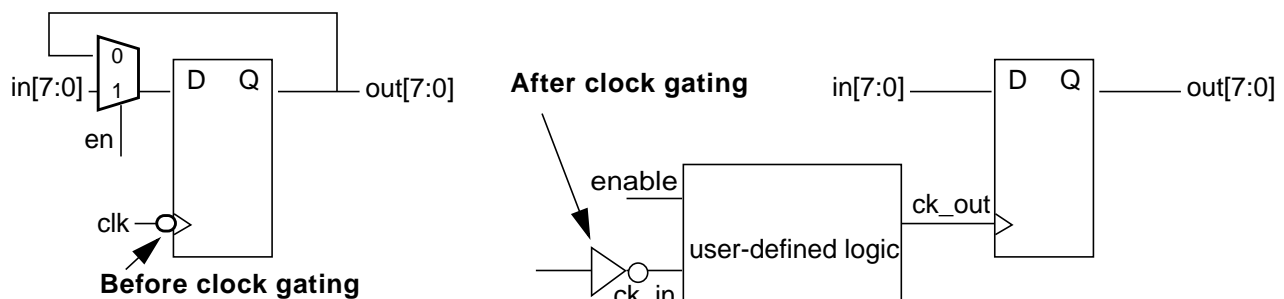
By only creating the clock-gating logic for positive-edge triggered registers, the QOS might be affected. In this case, all gated flip-flops are mapped to positive-edge triggered flip-flops, even if they are specified as negative-edge triggered flip-flops in the RTL and even if the library has negative-edge triggered flip-flops available.

Example 6-5 Clock-Gating if Only Clock-Gating Module for Positive-Edge Triggered Registers

Figure 6-12 shows the schematic before and after clock-gating insertion for the sample RTL code shown below. Note that the RC-LP engine inserts an inverter at the clock input of the inserted clock-gating logic to adapt the logic (which was created for a positive-edge triggered logic) for the original negative edge-triggered register.

```
module ex1 (in, out, en, clk);
  input en, clk;
  input [7:0] in;
  output [7:0] out;
  reg [7:0] out;
  always @ (negedge clk)
  begin
    if (en)
      out <= in;
  end
endmodule
```

Figure 6-12 Circuit with Negative Edge-Triggered Logic before and after Clock Gating



Example 6-6 Clock Gating with Clock-Gating Modules for Positive and Negative-Edge Triggered Registers

Figure 6-13 on page 120 shows the RTL description of the `wrapper.v` file which contains a clock-gating module for positive and negative-edge triggered registers.

Figure 6-14 on page 121 shows the schematic of a clock-gated design using the user-created clock-gating modules. In this case only the clock-gating logic for the positive-edge triggered registers was needed.

Figure 6-15 on page 121 shows the flow used to support the use of a user-defined clock-gating module.

Figure 6-13 Module `wrapper.v`

```
module my	CG_MOD (ck_in, enable, test, ck_out);
input ck_in, enable, test;
output ck_out;
wire tm_out, ck_inb;
reg enl;

assign tm_out = enable | test;
assign ck_inb = ~ck_in;
always @ (ck_inb or tm_out)
    if (ck_inb)
        enl = tm_out;
assign ck_out = ck_in & enl;
endmodule

module my	CG_MOD_neg (ck_in, enable, test, ck_out);
input ck_in, enable, test;
output ck_out;
wire tm_out;
reg enl;

assign tm_out = enable | test;
always @ (ck_in or tm_out )
    if (ck_in)
        enl = tm_out;

assign ck_out = ck_in | ~enl;
endmodule
```


Figure 6-14 Clock-Gated Design Using User-Created Clock-Gating Module

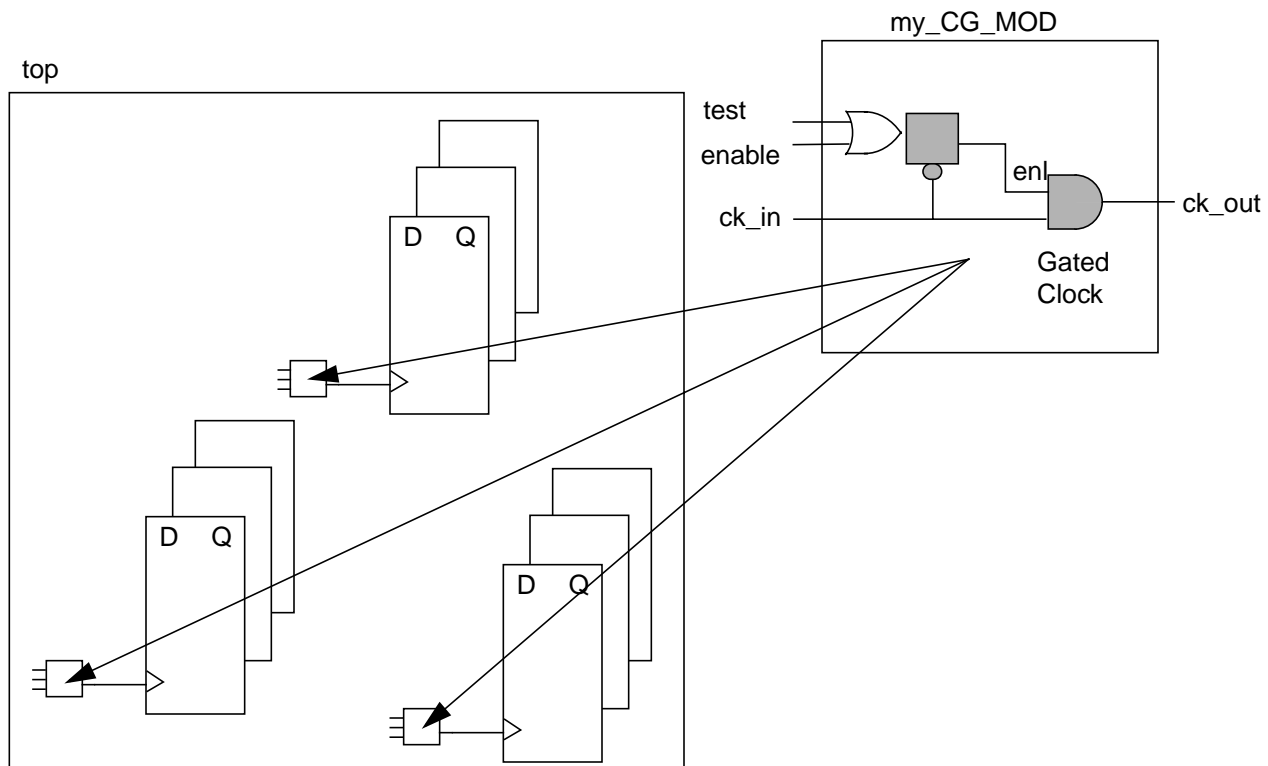


Figure 6-15 Flow Supporting User-Created Clock-Gating Module

```
# General set up
set_attribute lib_search_path ..
set_attribute hdl_search_path ..

# Enable clock-gating insertion
set_attribute lp_insert_clock_gating true

# Read the target libraries
set_attribute library my.lib

# Load the design and the clock-gating wrapper module
read_hdl design.v wrapper.v

# Elaborate the design and the clock-gating wrapper module
elaborate

# Set the timing and design constraints on the top module of the design
read_sdc design.sdc

# Specify the clock-gating module for the positive-edge triggered registers
set_attribute lp_clock_gating_module my_CG_MOD /designs/design

# Map to cells of technology library and optimize design on top module of the design
synthesize -to_mapped

# Report clock-gating results
report clock_gating

write -m > clock_gated.v
```

Using Attributes to Select an Integrated Clock-Gating Cell in the Technology Library

If no specific library cell is specified through the `lp_clock_gating_cell` attribute, and no clock-gating module is specified through the `lp_clock_gating_module` attribute, the RC-LP engine tries to find an integrated clock-gating cell in the technology library. Integrated clock-gating cells are identified in the library by the `clock_gating_integrated_cell` attribute.

```
clock_gating_integrated_cell : {generic|string};
```

When you specify `generic` as the value, the RC-LP engine determines the actual functionality from the state table of the cell. When you specify `string` as the value, the string must be a concatenation of up to four strings specifying

- The first string specifies the type of sequential element. Possible values are: `latch`, `flip-flop`, or `none`.
- The second string specifies whether the logic is appropriate for positive or negative edge-triggered registers. Possible values are: `posedge` or `negedge`.
- The third (optional) string specifies whether the test-control logic is located before or after the latch or flip-flop, or does not exist. For a latch or flip-flop, possible values are: `precontrol`, `postcontrol`, or no entry. If you use no sequential element in the gating logic, possible values are: `control` or no entry.
- The fourth (optional) string, which exists only if the third string does, specifies whether you want observability logic or not. Possible values are: `obs` or no entry.

For more information about the `clock_gating_integrated_cell` attribute, see the [Clock-Gating Cell Specification](#) in the *Library Guide for Encounter RTL Compiler*.

The technology library can have up to 26 different types of integrated clock-gating cells. The RC-LP engine supports all 26 types.

- To indicate which integrated clock-gating cell the RC-LP engine should use in the design, set the following clock-gating directives (design attributes) in RTL Compiler:

- ❑ `lp_clock_gating_style` selects the type of the sequential element.

The default is `latch`, indicating that the integrated clock-gating cell contains a latch. Other values are `none` and `ff`.

For more information on the impact of this selection, see [Choosing the Correct Clock-Gating Logic](#).

- ❑ `lp_clock_gating_add_reset` determines whether the integrated clock-gating cell must contain asynchronous reset logic added to the glitch removing logic.

Low Power in Encounter RTL Compiler

Clock Gating

The default is `false`, indicating that no asynchronous reset logic is added.

- ❑ `lp_clock_gating_control_point` controls whether the integrated clock-gating cell contains test-control logic, and specifies where it must be located.

The default is `precontrol`, indicating that test-control logic is inserted before the sequential element. Other values are `none` and `postcontrol`. For more information refer to [Selecting Clock-Gating Logic with Control Logic](#).

- ❑ `lp_clock_gating_add_obs_port` specifies whether the integrated clock-gating cell contains observability logic or not.

The default is `false`, indicating that no observability logic is added. For more information refer to [Selecting Clock-Gating Logic with Observability Logic](#).

By default, the RC-LP engine selects a cell that has `latch_posedge_precontrol` or `latch_negedge_precontrol` as a value for the `clock_gating_integrated_cell` attribute.

Important

If your library does not contain the integrated clock-gating cell you request through the attributes, the RC-LP engine will use regular library cells to instantiate the clock-gating module that corresponds to the clock-gating directives.

[Example 6-7](#) on page 127 illustrates clock gating using an integrated clock-gating cell, while [Example 6-8](#) on page 128 shows clock gating if the requested cell does not exist.

Choosing the Correct Clock-Gating Logic

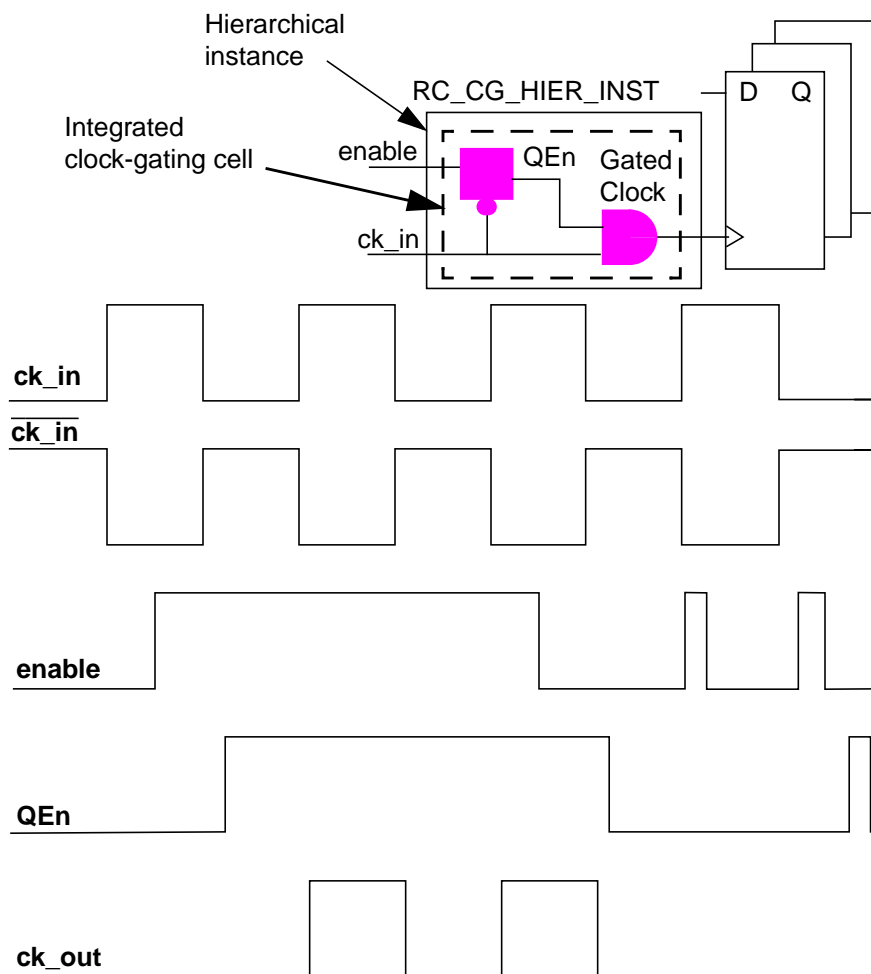
The type of integrated clock-gating cell determines how a glitch on the enable signal is handled.

By default, the RC-LP engine selects the glitch-free latch gating logic. Figure [6-16](#) shows an example of the latch-based clock-gating logic.

Low Power in Encounter RTL Compiler

Clock Gating

Figure 6-16 Impact of a Glitch on a Latched Clock Gate (latch_posedge)



To minimize the generation of glitches, the enable signal (enable) is generated from the opposite clock phase that triggers the registers. This gives the enable signal half a cycle for the glitch to die in the enable logic.

Low Power in Encounter RTL Compiler

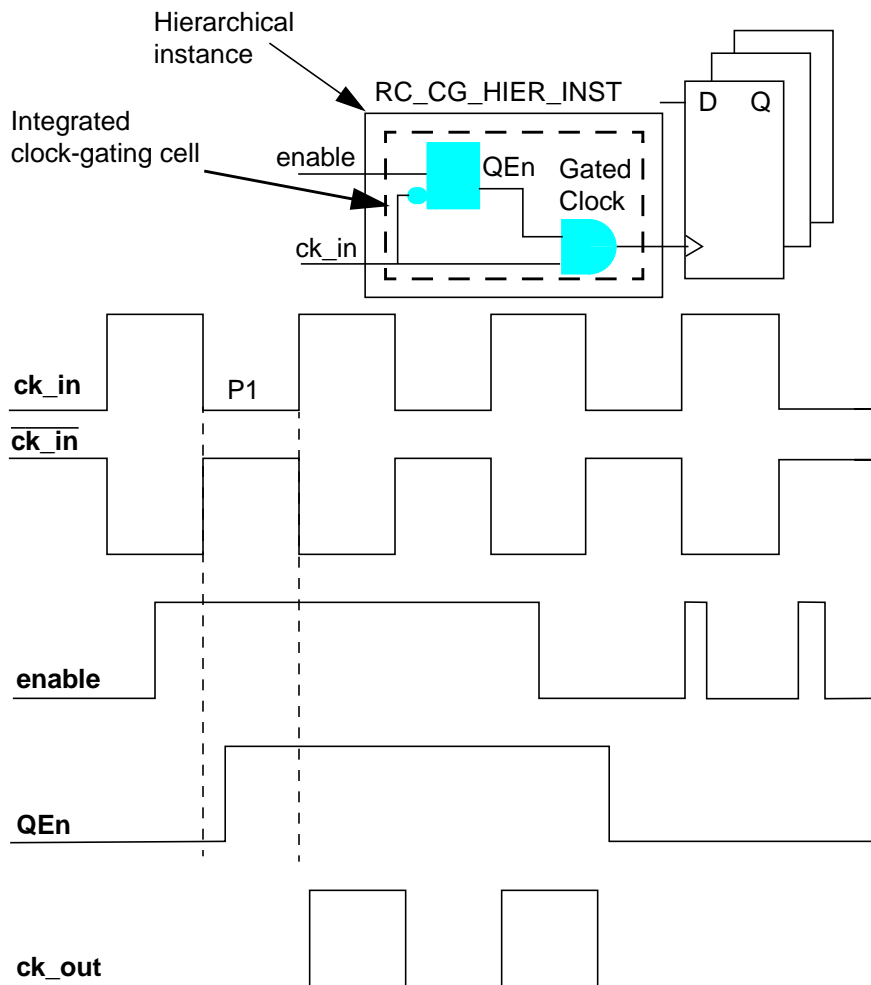
Clock Gating

To select a flip-flop-based clock-gating logic, set

```
set_attribute lp_clock_gating_style ff /designs/design
```

Figure 6-17 shows an example of the flip-flop-based clock-gating logic.

Figure 6-17 Impact of a Glitch on a Flip-Flop Gated Clock (ff_posedge)



Note: If the enable signal (`enable`) changes during period P1, the change of the enable signal cannot be captured for a flip-flop based gating style and thus can lead to an incorrect design. You might find a functional mismatch between the original RTL and the gated netlist when performing functional verification.

If the registers have an asynchronous reset (as defined in the RTL), you can set the `lp_clock_gating_add_reset` attribute to `true` to ensure that the glitch removal flip-flop has the correct initial value.

Low Power in Encounter RTL Compiler

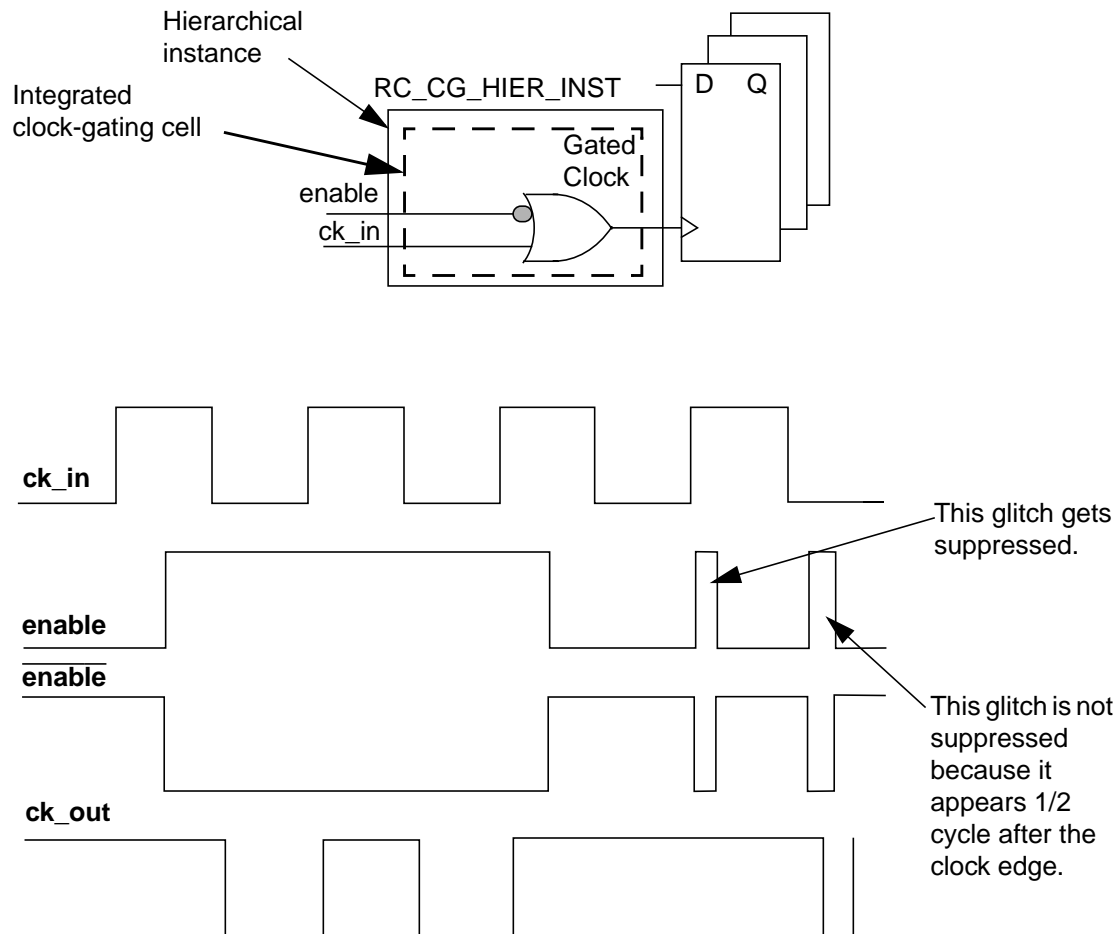
Clock Gating

To select a non-latch clock-gating logic, set

```
set_attribute lp_clock_gating_style none /design/design
```

Figure 6-18 shows an example of the none latched clock-gating logic. In this case, the glitch from the enable signal might propagate to the generated clock-gating pin.

Figure 6-18 Impact of a Glitch on a Non-Latched Clock Gate (none_posedge)



Note: If you want to change the setup timing check for the clock-gating logic, you can use the `path_adjust` command. For example,

```
path_adjust -to cgic_instance/En -delay 200
```

Example 6-7 Clock Gating Using Integrated Clock-Gating Cell

In this example, you want to use an integrated clock-gating cell for clock-gating insertion. The cell is selected based on the default values for the clock-gating directives:

- `lp_clock_gating_style` defaults to `latch`.
- `lp_clock_gating_add_reset` defaults to `false`.
- `lp_clock_gating_control_point` defaults to `precontrol`.
- `lp_clock_gating_add_obs_port` defaults to `false`.

In this case, the technology library contains the requested cell as is illustrated in the output generated by RTL Compiler.

RTL Code

```
module top (enable, in1, in2, clk, out1, out2);
input enable, clk;
input [7:0] in1,in2;
output [7:0] out1, out2;
reg [7:0] out1, out2;

always @(posedge clk)
    if (enable)
        out1 = in1 + in2;

always @(negedge clk)
    if (enable)
        out2 = in1 * in2;

endmodule
```

Script

```
# General setup.
set_attribute information_level 4 /
set_attr lib_search_path /lib/tech/tsmc_13/

# Enable clock-gating insertion
set_attr lp_insert_clock_gating true

# Read in the technology library
set_attr library mylib.lib

# Read in design and constraints
load top.v
elaborate
read_sdc top.sdc

# Map cells, insert clock-gating logic, optimize the design
synthesize -to_map

# Report results
report clock_gating -detail

write -m > top.mv
```

Low Power in Encounter RTL Compiler

Clock Gating

Output

```
...
Mapping top to gates.
  Mapping 'top'...
    Preparing the circuit
    Gating clocks
Info    : Could not find any user created clock-gating module. [POPT-12]
        : Looking for Integrated clock-gating cell in library.
Clock Gating Status
=====
Category                                Number      Percentage
-----
Gated flip-flops                        16          100%
Ungated flip-flops
  Excluded from clock-gating            0           0%
  Enable signal is constant              0           0%
  Register bank width too small          0           0%
Cannot map to requested logic          0           0%
Total flip-flops                        16          100%
Total CG Modules                         2
    Done preparing the circuit
...
```

The output shows that

- No user created clock-gating module was found, indicating that the RC-LP engine will look for an integrated clock-gating cell
- The requested integrated clock-gating cell was found.

No attributes in the script were set to select the clock-gating cell, so the RC-LP engine used the default values and selected a `latch_posedge_precontrol` cell which was available in the library.

Example 6-8 Clock Gating Using Discrete Clock-Gating Logic

In this example, you use the same RTL code and a similar script as for [Example 6-7](#) on page 127. You only use a different technology library. In this case, the library does not contain the requested cell.

Script

```
# General setup.
set_attribute information_level 4 /
set_attr lib_search_path /lib/tech/tsmc_13/
# Enable clock-gating insertion
set_attr lp_insert_clock_gating true
# Read in the technology library
set_attr library mylib_no_cgic.lib
# Read in design and constraints
load top.v
elaborate
read_sdc top.sdc
```


Low Power in Encounter RTL Compiler

Clock Gating

```
# Map cells, insert clock-gating logic, optimize the design
synthesize -to_map

# Report results
report clock_gating -detail

write -m > top.mv
```

Output

```
...
      Gating clocks
Info    : Could not find any user created clock-gating module. [POPT-12]
        : Looking for Integrated clock-gating cell in library.
Info    : Cannot find requested type of clock-gating integrated cell. [POPT-10]
        : The library has no integrated clock-gating cells of type
'Latch_Posedge_Precontrol' and 'Latch_Negedge_Precontrol'.
Info    : Created discrete clock-gating module. [CG-103]
        : RC_CG_MOD_AUTO_top
        : Two discrete clock-gating modules are created: one for the positive-edge
        : triggered registers and one for the negative-edge triggered registers.
        : The names of the clock-gating modules are based on the name of the design.
Info    : Created discrete clock-gating module. [CG-103]
        : RC_CG_MOD_AUTO_top_neg
...
```

The output shows that

- No user created clock-gating module was found, indicating that the RC-LP engine will look for an integrated clock-gating cell
- The requested integrated clock-gating cell was not found.
- Two discrete clock-gating modules are created: one for the positive-edge triggered registers and one for the negative-edge triggered registers. The created clock-gating module is `RC_CG_MOD_AUTO_top`.

To check the components used to create the discrete logic, use the following command:

```
rc:/>write_hdl RC_CG_MOD_AUTO_top > RC_CG.vm
```

Figure 6-19 shows the netlist of the discrete clock-gating module. Three standard components from the `mylib_no_cgic.lib` were used to create the logic.

Figure 6-19 Netlist of Discrete Clock-gating Module

```
// Generated by Cadence RTL Compiler-D (RC) version

module RC_CG_MOD_AUTO_top(ck_in, enable, test, ck_out);
  input ck_in, enable, test;
  output ck_out;
  wire ck_in, enable, test;
  wire ck_out;
  wire en1, n_2;
  AND2X2 g10(.A (ck_in), .B (en1), .Y (ck_out));
  TLATNX1 en1_reg(.D (n_2), .GN (ck_in), .Q (en1), .QN ());
  OR2X1 g12(.A (enable), .B (test), .Y (n_2));
endmodule
```

Controlling Insertion of Clock-Gating Logic

The RC-LP engine has the flexibility to let you control how clock gating is implemented.

Preventing Clock Gating

- To prevent clock gating from being added to the entire design, a subdesign or an instance (of type register), set the following attribute to the appropriate object:

```
set_attribute lp_clock_gating_exclude true /designs/design
set_attribute lp_clock_gating_exclude true /designs/subdesigns/subdesign
set_attribute lp_clock_gating_exclude true /des*/design/instances_seq/name
```

Note: You can only apply the `lp_clock_gating_exclude` attribute to a *unique* hierarchical or flat instance. Otherwise, the tool issues an error message and the attribute is ignored. If you want to exclude the subdesign or instance from clock gating, first uniquify it using the `edit_netlist_dedicate_subdesign` command before setting the `lp_clock_gating_exclude` attribute.

- To prevent clock gating of an instance (flip-flop) when intending to perform *retiming* on the design, you must in addition prevent retiming of that instance. Set these attributes:

```
set_attribute dont_retime true [find / -instance name]
set_attribute lp_clock_gating_exclude true /des*/design/instances_seq/name
```

Controlling the Fanout of the Clock-Gating Logic

- To determine the minimum number of registers required to enable clock-gating insertion, set the following design attribute:

```
set_attribute lp_clock_gating_min_flops integer /designs/design
```

The default is 3.

- To determine the maximum number of registers that can be driven by each clock-gating instance, set the following design attribute:

```
set_attribute lp_clock_gating_max_flops integer /designs/design
```

The default is 2048.

If a register bank has a bit width larger than the specified size, the RC-LP engine will duplicate the clock-gating cells and distribute the registers evenly over the clock-gating cells. For example, if the register bank width is 32, and the maximum number is 20, each clock-gating cell will be driving 16 registers.

Note: By default the RC-LP engine will also gate any orphan flip-flops.

Note: Currently, the RC-LP engine does not respect the `max_fanout` attributes during clock gating.

Controlling Clock Gating Across the Hierarchy

By default, the RC-LP engine inserts clock-gating instances that can gate registers located in different hierarchical instances. As a result, a port for the gated clock can be added to the hierarchical instances.

To prevent the addition of ports to the hierarchical instances you can set either of the following attributes:

```
set_attribute hard_region true {subdesign|instance}  
set_attribute boundary_opto false subdesign
```

As a result the clock-gating instances can only gate registers within the hierarchical instance in which they are inserted.



Tip

In addition, if the setting of the lp_clock_gating_min_flops attribute can not be satisfied, no clock gating will be inserted.

Extracting Common Enable

Register banks can sometimes not be considered for clock-gating insertion because their bit width is smaller than the minimum number of registers required for clock-gating insertion. If the flops that were not gated—because of this minimum bit width requirement—have a complex enable logic, common enable extraction can extract a common function in the enable logic of those registers. By considering this common function as the enable signal, the minimum bit width requirement can be satisfied and the registers can be gated.

- To enable common enable extraction, set the following design attribute:

```
set_attribute lp_clock_gating_extract_common_enable true /designs/design
```

By default, this attribute is not enabled.

Consider the design in [Figure 6-20](#) on page 132. The design has two register banks with width 2. With the default setting of the lp_clock_gating_min_flops attribute and with the lp_clock_gating_extract_common_enable attribute disabled, these registers cannot be considered for clock-gating insertion.

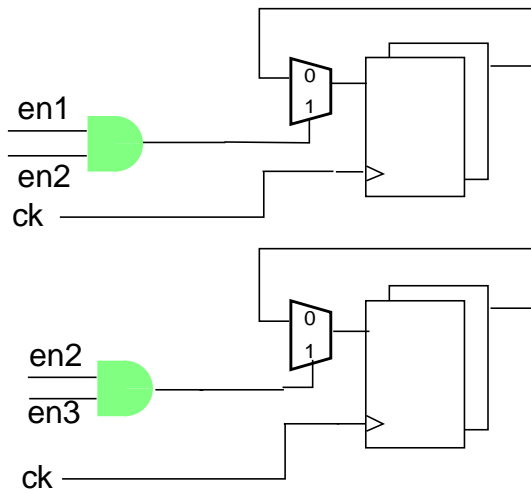
However, both register banks have two enable signals each and they have one enable signal in common. By enabling the lp_clock_gating_extract_common_enable attribute, the common enable signal `en2` can be considered as the enable signal for the clock-gating logic and because this signal now controls four registers which is larger than the minimum requirement, clock gating can be inserted as shown on the right side of [Figure 6-20](#) on page 132.

Low Power in Encounter RTL Compiler

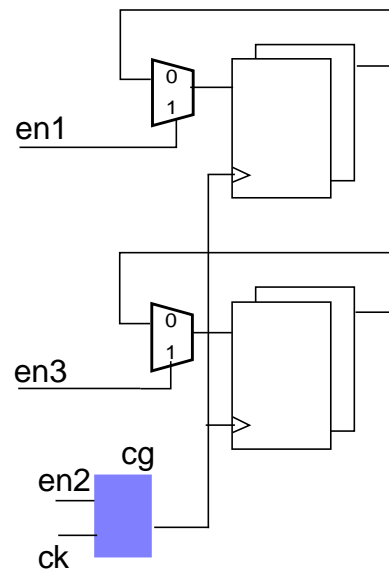
Clock Gating

Figure 6-20 Clock Gating with Extracted Common Enable

a. before common enable extraction



b. after common enable extraction



Previewing the Effect of Clock Gating

After elaborating the design, you can check the effect that clock gating might have on the design. Early exploration allows you to make key adjustments early in the design cycle.

- To run a clock-gating what-if analysis, use the `report_clock_gating` command with the `-preview` option.

```
report_clock_gating -preview [-gated_ff]
                        [-clock clock_list | -clock_pin {pin|port|subport}...]
```

To perform a clock-gating what-if analysis, RTL Compiler requires that you have defined the clocks either with the `define_clock` command or through an SDC constraint. If you did not define any clocks you must specify the clock inputs for which you want to perform the what-if analysis using the `-clock_pin` option.

Note: If both `-clock` and `-clock_pin` options are specified, the `-clock` option takes precedence.

The `-gated_ff` option adds the names of the flip-flops for which clock-gating logic would be inserted.

You can use this feature with unmapped and partially mapped netlists. Currently, the RC-LP engine only explores the generic portions of a partially mapped netlist, and reports already gated flops in the partially mapped netlist.

Exploration is affected by the following attributes that control clock-gating insertion:

- `lp_clock_gating_exclude`
- `lp_clock_gating_min_flops`
- `lp_clock_gating_max_flops`
- `hard_region`
- `boundary_opto`

For more information on these attributes, refer to [Controlling Insertion of Clock-Gating Logic](#).

In addition, for multiple supply multiple voltage (MSMV) designs, the following attributes are taken into account:

- `power_domain`
- `shutoff_signal`
- `shutoff_signal_polarity`

Low Power in Encounter RTL Compiler

Clock Gating

Example 6-9 shows a what-if analysis report for clock gating. The higher the reduction is for the clock toggle, the higher the potential dynamic power savings.

Example 6-9 Clock-Gating What-If

```
rc:/> report clock_gating -preview -gated_ff -clock_pin clk
Info: Since -preview option is specified, options other than -clock, -clock_pin,
-gated_ff and -ungated_ff are ignored
  Previewing clock-gating logic in netlist from '/designs/top'

=====
CG preview for '/designs/top/ports_in/clk'
=====
Number of flops with this clock input : 9
Number of flops that could be clock gated : 8

Enable : /designs/top/instances_hier/i2/instances_comb/g1/pins_out/z
  Number of flops with these clock and enable inputs : 4
  Number of flops that could be clock gated          : 4
  Enable switching activity                          : (0.31620 0.020000)
  Reduction in clock toggle (in percent)             : 18.26%
  Gated flops -->
    i2/out_reg[3]
    i2/out_reg[0]
    i2/out_reg[1]
    i2/out_reg[2]

Enable : /designs/top/instances_hier/i1/instances_comb/g1/pins_out/z
  Number of flops with these clock and enable inputs : 4
  Number of flops that could be clock gated          : 4
  Enable switching activity                          : (0.30620 0.015000)
  Reduction in clock toggle (in percent)             : 31.79%
  Gated flops -->
    i1/out_reg[0]
    i1/out_reg[1]
    i1/out_reg[2]
    i1/out_reg[3]
```

Clock Gating with DFT

Any kind of testing requires applying a known input stimulus to the input pins of the design to be tested. This input stimulus can be applied using a simulator, a tester, or other logic within the system. Having the ability to apply the stimulus to the design under test is generally referred to as *controllability*. The ability to evaluate the output response of the design under test is generally referred to as *observability*.

Depending on the type of clock-gating logic that you insert in your design, the gated clock net often can no longer be controlled during test synthesis, resulting in a potential reduced fault coverage for the design. Also, the enable signal driving the control logic often can no longer be observed.

You can ensure that the clock-gating logic that is inserted by the low power (RC-LP) engine is controllable and observable by following these steps:

- [Selecting Clock-Gating Logic with Control Logic](#)
- [Specifying the Control Signals](#)
- [Connecting the Test Signals](#)
- [Selecting Clock-Gating Logic with Observability Logic](#)
- [Inserting the Observability Logic](#)

The extra steps required to combine design for test and clock-gating are reflected in the [Recommended Top-Down Clock Gating Flow with DFT](#). For more information on the bottom-up flow, refer to [Recommended Bottom-Up Clock Gating Flow with DFT](#).

Selecting Clock-Gating Logic with Control Logic

- To make sure that the gated clock signal is controllable, do one of the following:
 - ❑ Create your own clock-gating module with controllability logic.
 - ❑ Select a clock-gating cell that contains test-control logic by setting the following design attribute:

```
set_attribute lp_clock_gating_control_point {precontrol|postcontrol} \  
/designs/design
```

If your library does not contain an integrated clock-gating cell that contains the requested test-control logic, the RC-LP engine will use regular library cells to instantiate the clock-gating module that corresponds to this clock-gating directive.

Note: If you set this attribute to `none`, you select a clock-gating cell without test-control logic.

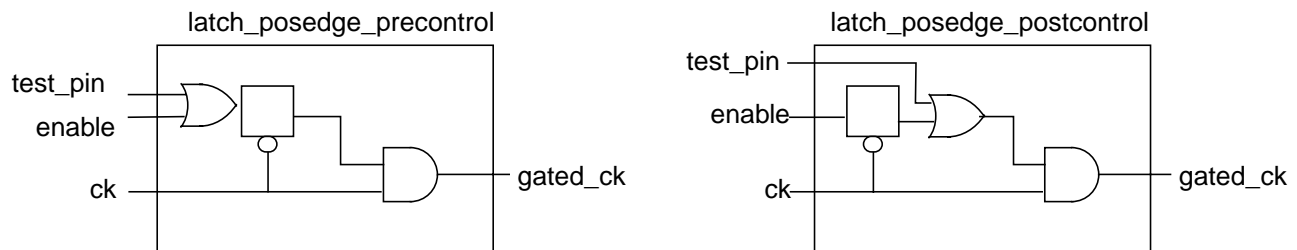
Low Power in Encounter RTL Compiler

Clock Gating

Figure 6-21 on page 136, shows the difference between `precontrol` and `postcontrol` in a latch-based integrated clock-gating cell for positive edge-triggered latches:

- `precontrol` indicates that the test-control logic is located before the latch
- `postcontrol` indicates that the test-control logic is located after the latch

Figure 6-21 Possible Locations of the Test-Control Logic



Specifying the Control Signals

The test pins of the clock-gating instances must be connected to a test-control signal. You can indicate to use a *specific* test-control signal to connect to the test pins of either all clock-gating instances in the design or of specific clock-gating instances. You can also choose to use the shift-enable signal of the scan chain to which the flip-flops gated by the clock-gating instances belong. You can further select to use this method for the entire design or for specific clock-gating instances.

- To specify the test-control signal to use for *all* clock-gating instances in the design, either
 - ❑ Specify to use a specific test-control signal

```
set_att lp_clock_gating test_signal test_signal_path /de*/design
```

Applies the same (specified) signal for all clock-gating instances.
 - ❑ Specify to use the shift-enable signal

```
set_att lp_clock_gating test_signal use_shift_enable /de*/design
```

If the design uses several shift-enable signals, multiple test-control signals will be used to connect to the test pins of the clock-gating instances.

Important

If you want to use the shift-enable signal as the test-control signal, you should set this attribute after you have connected the scan chains.

Low Power in Encounter RTL Compiler

Clock Gating

- To specify the test-control signal to use for *specific* clock-gating instances, either

- ❑ Specify to use a specific test-control signal

```
set_att lp_clock_gating_test_signal test_signal_path \  
cg_instance1 cg_instance2 ..
```

Applies the same (specified) signal for the specified clock-gating instances.

- ❑ Specify to use the shift-enable signal

```
set_att lp_clock_gating_test_signal use_shift_enable \  
cg_instance1 cg_instance2 ...
```

If different shift-enable signals are applied to the scan flops gated by the specified clock-gating instances, multiple test-control signals will be used to connect to the test pins of the clock-gating instances.

If you set the `lp_clock_gating_test_signal` attribute only on specific clock-gating instances (and not on the design), RTL Compiler will tie the test pin of those clock gating instances for which you did not specify a test signal to the inactive value. For example, for active high (low) test pin, the pin is tied to constant 0 (1). By default, the test pins of the integrated clock-gating cells are active high, unless you changed the value of the `polarity` attribute to false. For more information, see the [Clock-Gating Cell Specification](#) in the *Library Guide for Encounter RTL Compiler*.

The `test_signal_path` is the RC path to the test signal. Test signals are defined using the `define_dft test_mode` or the `define_dft shift_enable` command and are stored at `/designs/top_design/dft/test_signals`.



Tip

When defining the test signal to use for clock gating, specify the test signal using the `-name` option of the `define_dft test_mode` or the `define_dft shift_enable` constraints. This approach allows you to pass a user-defined name to the `lp_clock_gating_test_signal` attribute instead of a tool-generated hierarchical reference name.

```
define_dft test_mode -name test_signal_name -active high test_mode_port
```

Connecting the Test Signals

There are two scenarios to connect the test pins of the clock-gating logic:

- [Using Auto-Connection Capability](#)
- [Using Post-Synthesis Process Capability](#)

Using Auto-Connection Capability

If you select to use a specific test-control signal for all clock-gating instances in the design, and you set the `lp_clock_gating_test_signal` design attribute before you start synthesizing, the RC-LP engine can connect the test-control signal to the test pin of the clock-gating logic during clock-gating insertion.

This scenario would apply if you or your team handle all aspects of the design, such as clock-gating insertion and scan insertion in the same flow. This is shown in the [Recommended Top-Down Clock Gating Flow with DFT](#) and the [Recommended Bottom-Up Clock Gating Flow with DFT](#).

Using Post-Synthesis Process Capability

If all other cases, you need to use the following command to connect the test pins:

```
clock_gating connect_test
```

If you select to use a specific test-control signal for *specific* clock-gating instances, you can specify this command right after mapping the design.

If you select to use the shift-enable signal, you must specify this command after you connect the scan chains.

You would also need to use this command when different teams handle different aspects of the design. For example, one team might perform scan insertion after another team inserted clock gating and performed timing optimization. This flow is illustrated in [Scan Insertion after Clock-Gating Insertion](#).

Selecting Clock-Gating Logic with Observability Logic

- To make sure that the enable signal of the clock-gating logic is observable, either
 - ❑ Ensure that the enable signal of the clock-gating logic in your clock-gating module is observable—your clock-gating module must have an `obs` port.
 - ❑ Select a clock-gating cell that contains observability logic by setting the following design attribute:

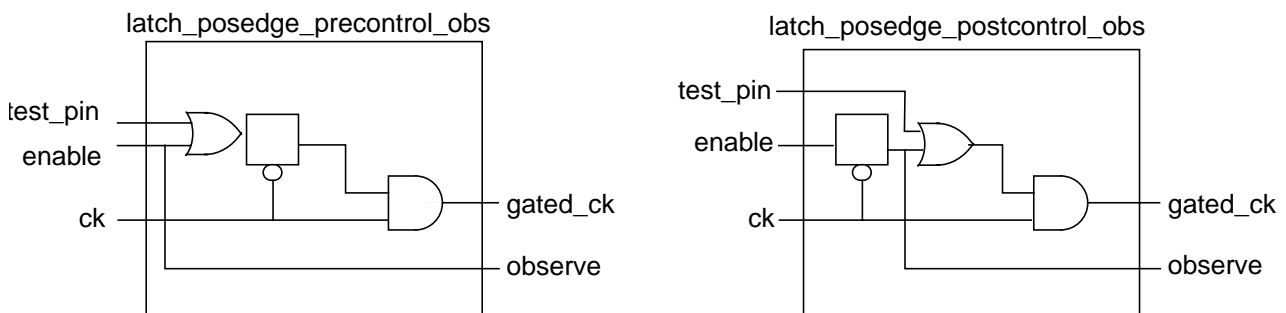
```
set_attribute lp_clock_gating_add_obs_port true /designs/design
```

If your library does not contain an integrated clock-gating cell that contains the requested observability logic, the RC-LP engine will use regular library cells to instantiate the clock-gating module that corresponds to this clock-gating directive.

Note: If you set this attribute to `false`, the RC-LP engine selects a clock-gating cell without observability logic. Consequently, the clock-gating module will not have an `obs` port.

Figure 6-22 shows the clock-gating cells shown in [Figure 6-21](#) on page 136, but with observability logic added.

Figure 6-22 Clock-Gating Cells with Observability Logic



Inserting the Observability Logic

If you set the `lp_clock_gating_add_obs_port` attribute to `true`, the RC-LP engine creates an observation port in the clock-gating module, but you still need to insert the observability logic.

- To insert the observability logic, enter the following command *after* the clock-gating has been inserted:

```
clock_gating insert_obs [-hierarchical] [-max_cg integer]
```

Low Power in Encounter RTL Compiler

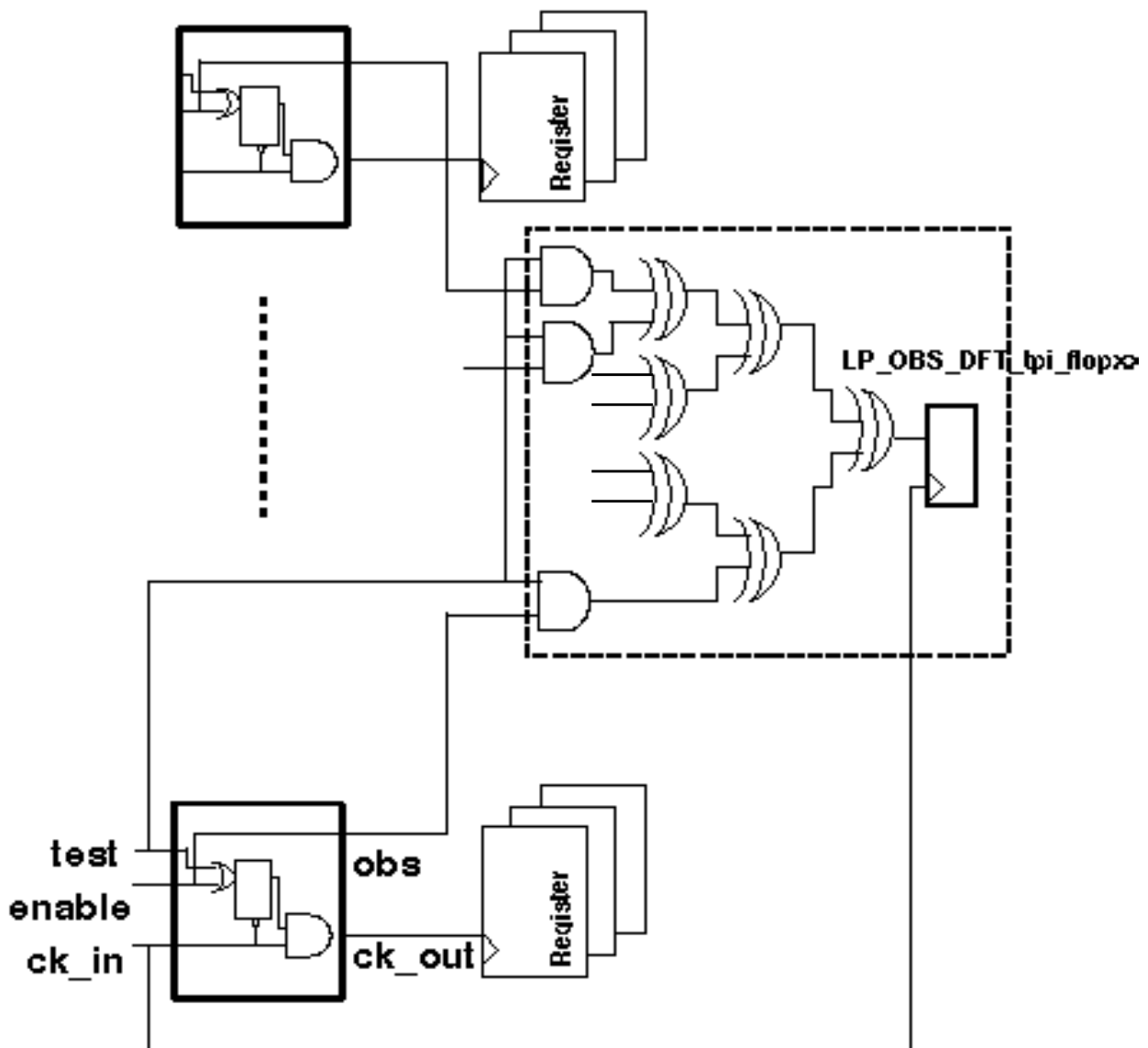
Clock Gating

This command allows you to

- Insert and connect observability logic across hierarchies using the `-hierarchical` option. Without this option, the RC-LP engine inserts observability logic in each module.
- Specify the maximum number of clock-gating cells that can be observed per observation flip-flop using the `-max_cg` option. By default, a maximum of 8 clock-gating cells can be observed per flop.

As shown in Figure 6-23, the RC-LP engine creates a balanced XOR-tree to share the observation flip-flop among the enable signals to be observed. Although the inserted logic is mapped, the observation flop—whose output is floating—still must be mapped to scan.

Figure 6-23 Example of Observability Logic

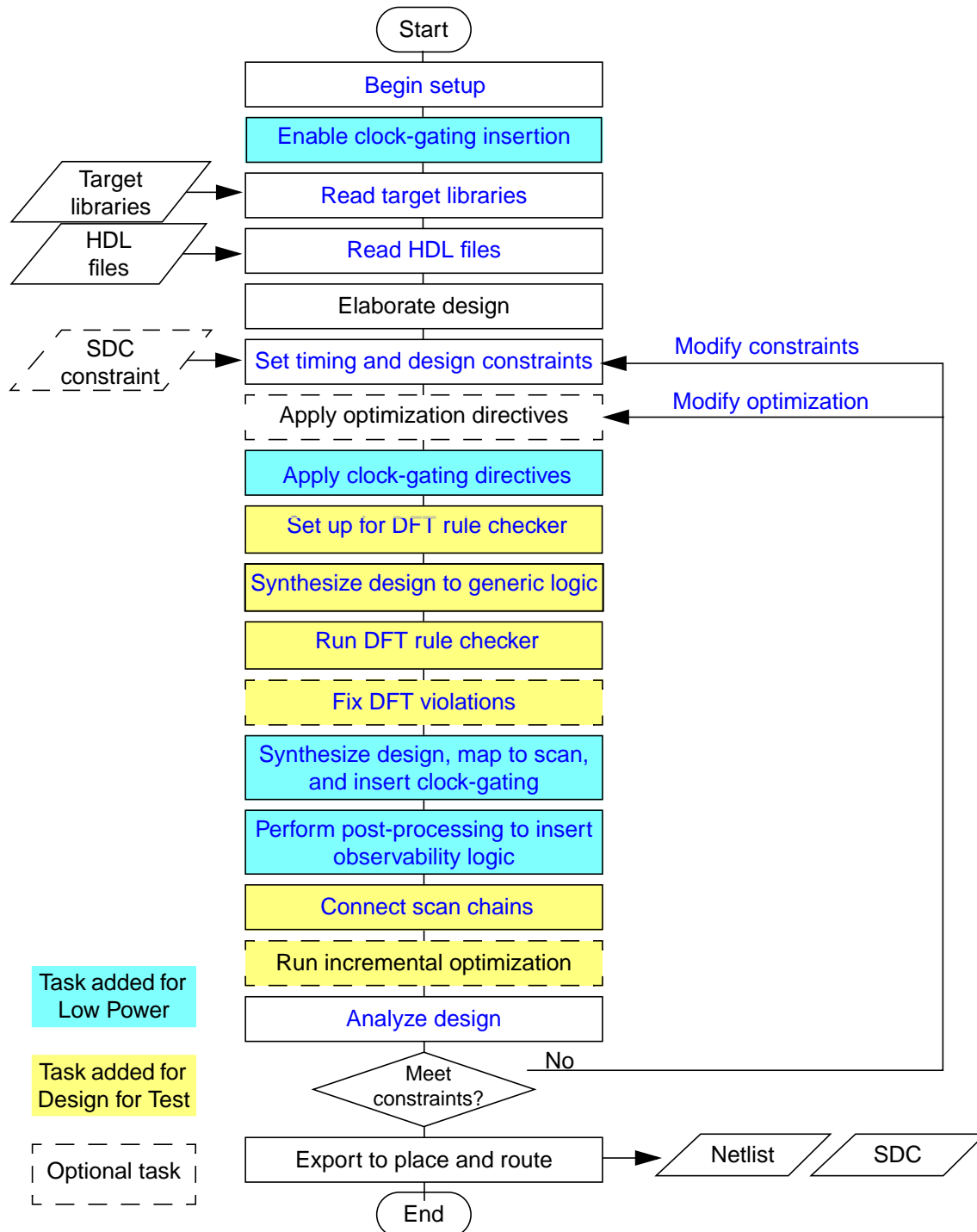


Low Power in Encounter RTL Compiler

Clock Gating

Recommended Top-Down Clock Gating Flow with DFT

This section briefly describes the steps in the flow which combines clock gating with DFT. The flowchart below shows the additional steps required in the low power flow.



Low Power in Encounter RTL Compiler

Clock Gating

Note: Additional steps for DFT are highlighted.

1. General set up.

```
set_attribute lib_search_path ...
set_attribute hdl_search_path ...
```

2. Enable clock-gating insertion.

```
set_attribute lp_insert_clock_gating true /
```

3. Read the target libraries.

```
set_attribute library library_list /
```

4. Read and elaborate the design.

```
read_hdl verilog_files
elaborate
```

5. Set the timing and design constraints.

6. (Optional) Specify attributes to select clock-gating cells and to control clock gating.

```
set_attribute lp_clock_gating_add_obs_port true /designs/design
set_attribute lp_clock_gating_add_reset value /designs/design
set_attribute lp_clock_gating_control_point {precontrol|postcontrol} \
    /designs/design
set_attribute lp_clock_gating_style value /designs/design
```

7. Specify the DFT setup.

```
define_dft shift_enable -name se_signal -active high se_port
define_dft test_mode -name test_signal_name -active high test_mode_port
```

8. Generate a generic netlist to remove any dont-care cells.

```
synthesize -to_generic
```

9. Run the DFT rule checker.

```
check_dft_rules
report dft_registers
```

10. Identify the test-control signal used for the clock-gating logic to the RC-LP engine.

```
set_attribute lp_clock_gating_test_signal test_signal_path \
    /designs/design
```

Specify a test-control signal that was already defined with either the `define_dft test_mode` or the `define_dft shift_enable` command.

11. (Optional) Fix DFT rule violations.

```
fix_dft_violations
```

12. Map to cells of the technology library and optimize the design.

```
synthesize -to_mapped
```

Note: At this time the RC-LP engine inserts the clock-gating logic and auto-connects the

Low Power in Encounter RTL Compiler

Clock Gating

test pins. All flip-flops passing the DFT rule checker are mapped to scan flip-flops.

13. Insert the observability logic for the clock-gating instances.

```
clock_gating insert_obs -hierarchical
```

14. Run the DFT rule checker, to determine the DFT status of the observability logic:

```
check_dft_rules
```

15. Connect scan chains.

```
connect_scan_chains -auto_create_chains
```

16. To fix any timing issues introduced by inserting the observability logic and by connecting the scan chains, run incremental optimization:

```
synthesize -incremental
```

17. Report clock-gating results.

```
report clock_gating
```

Sample Script

```
# setup
set_attribute lib_search_path ..
set_attribute hdl_search_path ..
# enable clock-gating insertion.
set_attribute lp_insert_clock_gating true /
# read libraries and design
set_attribute library my.lib
read_hdl design.v
elaborate
# set timing constraints
read_sdc design.sdc
# DFT setup and run DFT rule checker
define_dft_shift_enable -name sen -active high SE
define_dft_test_mode -name tm -active high TM
synthesize -to_generic
check_dft_rules
# Identify test mode signal to RC-LP
set_attribute lp_clock_gating_test_signal /des*/design/*/test_signals/tm \
    /designs/design
# fix DFT constraints
fix_dft_violations
# synthesize, insert clock-gating logic and map to scan
synthesize -to_mapped
# include steps to insert observability logic
clock_gating insert_obs -hierarchical
check_dft_rules
# connect chains
connect_scan_chains -auto_create_chains
# fix timing issues caused by scan chain connection.
synthesize -incr
# analysis
report clock_gating
```

Checking System Messages during Optimization

It is important to check the system messages during the synthesis session.

For example, the following messages indicate a normal process. Out of nine flip-flops, eight were gated, and one was not because its enable signal was constant.

```
Mapping alu to gates.
  Mapping 'alu'...
    Preparing the circuit
      ...
      Gating clocks
    ...
Clock Gating Status
=====
Category                                Number      Percentage
-----
Gated flip-flops                        8           89%
Ungated flip-flops
  Excluded from clock gating             0           0%
  Enable signal is constant              1          11%
  Register bank width too small          0           0
  Cannot map to requested logic          0           0%
Total flip-flops                        9          100%
....
```

Troubleshooting

Enable signal is constant

The number for *Enable signal is constant* in the Clock Gating Status is equal to the number of *Total flip-flops* as shown below.

```
Mapping alu to gates.
  Mapping 'alu'...
    Preparing the circuit
      ...
      Gating clocks
    ...
Clock Gating Status
=====
Category                                Number      Percentage
-----
Gated flip-flops                        0           0%
Ungated flip-flops
  Excluded from clock gating             0           0%
  Enable signal is constant              9          100%
  Register bank width too small          0           0%
  Cannot map to requested logic          0           0%
Total flip-flops                        9          100%
```

Check your script or command log. The following command was most probably specified after the `elaborate` command.

```
set_attribute lp_insert_clock_gating true
```

Change your script to set this attribute before the `elaborate` command.

Low Power in Encounter RTL Compiler

Clock Gating

Excluded from clock gating

You have a number of flip-flops excluded from clock gating, but you did not set the `lp_clock_gating_exclude` to `true` on the design, any subdesign or sequential instance.

If you set exceptions on flops or on any of its pins using SDC constraints, the flops are marked 'preserve' by RTL Compiler and the RC-LP engine does not gate them. In the Clock Gating Status report, these flops are taken into account in the "Excluded from gating" category.

Reporting Clock-Gating Information

To report clock-gating information for the design, use the `report_clock_gating` command. This command provides capabilities for

- [Reporting Summary Information](#)
- [Reporting Gated and Non-Gated Flip-Flops](#)
- [Reporting Detailed Information on Clock-Gating Instances](#)
- [Reporting Detailed Information](#)

The information returned by the `report_clock_gating` command depends on your current position in the design hierarchy. For example, if you are in the subdesigns directory, the command returns information for the design. If you are in a lower directory of the design hierarchy, the command returns information for the hierarchical instance this subdirectory belongs to.



Tip

If you use a user-defined clock-gating module, you need to change your current location in the hierarchy to the `design` directory, before you enter this command.

Reporting Summary Information

- To print a summary report of the clock gating inserted in the design, enter:

```
rc:/> report_clock_gating -summary
```

The command reports the number of leaf clock-gating instances, the number of flip-flops gated by RC, the number of flip-flops gated by other tools, and the number of ungated flip-flops. The return value of this command is the total number of clock-gating instances inserted in the design.

Summary

Category	Number	%	

RC Clock Gating Instances		1	-
Non-RC Clock Gating Instances		0	-

RC Gated Flip-flops		8	89
Non-RC Gated Flip-flops		0	0
Ungated Flip-flops	1	11	
Total Flip-flops	9	100	

1

Low Power in Encounter RTL Compiler

Clock Gating

Reporting Gated and Non-Gated Flip-Flops

- To report all the flip-flops that are clock gated and the clock-gating instances that gate the flip-flops, enter:

```
rc:/> report clock_gating -gated_ff
```

The return value of this command is the number of gated flip-flops (8).

```
...
=====
Gated Flip-flops
-----
Module   Clock Gating Instance   Fanout   Gated Flip-flops
-----
alu      RC_CG_HIER_INST           8        aluout_reg[0]
                                     aluout_reg[1]
                                     aluout_reg[2]
                                     aluout_reg[3]
                                     aluout_reg[4]
                                     aluout_reg[5]
                                     aluout_reg[6]
                                     aluout_reg[7]
-----
Total    1                      8
=====
```

8

- To report all the flip-flops that are not clock gated, enter:

```
rc:/> report clock_gating -ungated_ff
```

The return value of this command is the total number of flip-flops that are not gated in the design, which is 1.

```
...
=====
Ungated Flip-flops
-----
Flip-flop Excluded Module Instance
-----
zero_reg   false    alu      alu
-----
Total ungated flip-flops: 1
-----
```

1

Reporting Detailed Information on Clock-Gating Instances

- To get more detailed clock-gating information on an inserted clock-gating instance (the hierarchical instance with the clock-gating logic inside), enter:

```
rc:/> report clock_gating -cg_instance name
```

Low Power in Encounter RTL Compiler

Clock Gating

Clock Gating Based on User-Defined Module

The following clock-gating report gives detailed information on an inserted clock-gating instance for [Example 6-6](#) on page 120.

```
rc:/> cd /designs/top
rc:/design/top> report clock_gating -cg_instance RC_CG_HIER_INST
Info: Since -cg_instance option is specified, all other options are ignored.
=====
...
=====
...
Clock Gating Instance : RC_CG_HIER_INST
-----
Origin:                Inserted by RC
Libcell:               none (User created clock-gating logic)
Module:                top (top)
Type:                  Leaf level CG Instance
Inputs:
  clk_in               =   clk (/designs/top/ports_in/clk)
                        TCF = (0.50000, 0.333333/ns)
  enable               =   enable (/designs/top/ports_in/enable)
                        TCF = (0.50000, 0.020000/ns)
  test                 =   LOGIC0
Outputs:
  clk_out              =   rc_gclk_146
                        TCF = (0.23750, 0.192500/ns)
Gated FFs:
Module  Clock Gating Instance  Fanout  Gated Flip-flops
-----
top     RC_CG_HIER_INST        8       out1_reg[0]
                                     out1_reg[1]
                                     out1_reg[2]
                                     out1_reg[3]
                                     out1_reg[4]
                                     out1_reg[5]
                                     out1_reg[6]
                                     out1_reg[7]
-----
Total   1                      8
=====
```

1

This report returns

- none for the library cell, indicating that a user-defined clock-gating module was used
- The name of the module (top)
- The nets connected to the inputs and outputs of the gating logic (clk, enable, and rc_gclk_146)
- The switching activities for the clock input (non-gated clock), enable, and clock output (gated-clock) pins
- The instance names of the flip-flops gated by this gating cell

Low Power in Encounter RTL Compiler

Clock Gating

Clock Gating Using Integrated Clock-Gating Cell

The following clock-gating report gives detailed information on an inserted clock-gating instance for [Example 6-7](#) on page 127.

```
rc:/> report clock_gating -cg_instance RC_CG_HIER_INST
Info: Since -cg_instance option is specified, all other options are ignored.
=====
...
=====
...
Clock Gating Instance : RC_CG_HIER_INST
-----
Origin:                Inserted by RC
Libcell:               TLATNTSCAX12 (mylib)
Style:                 latch_posedge_precontrol
Module:                top (top)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      clk (/designs/top/ports_in/clk)
                        TCF = (0.50000, 0.333333/ns)
  enable               =      enable (/designs/top/ports_in/enable)
                        TCF = (0.50000, 0.020000/ns)
  test                 =      LOGIC0
Outputs:
  ck_out               =      rc_gclk_148
                        TCF = (0.22880, 0.195000/ns)
Gated FFs:
Module  Clock Gating Instance  Fanout  Gated Flip-flops
-----
top     RC_CG_HIER_INST        8       out1_reg[0]
                                     out1_reg[1]
                                     out1_reg[2]
                                     out1_reg[3]
                                     out1_reg[4]
                                     out1_reg[5]
                                     out1_reg[6]
                                     out1_reg[7]
-----
Total   1                      8
=====
```

1

This report shows

- The library cell used for the clock-gating cell (TLATNTSCAX12) and the name of the library (mylib)
- The clock-gating style (latch_posedge_precontrol) which is the value that the TLATNTSCAX12 cell has for the clock_gating_integrated_cell attribute
- The switching activities for the clock input (non-gated clock), enable, and clock output (gated-clock) pins
- The instance names of the flip-flops gated by this gating cell

Low Power in Encounter RTL Compiler

Clock Gating

Clock Gating Using Discrete Clock-Gating Logic

The following clock-gating report gives detailed information on an inserted clock-gating instance for [Example 6-8](#) on page 128.

```
rc:/> report clock_gating -cg_instance RC_CG_HIER_INST
Info: Since -cg_instance option is specified, all other options are ignored.
=====
...
=====
...
Clock Gating Instance : RC_CG_HIER_INST
-----
Origin:                Inserted by RC
Libcell:               none (RC-LP created clock-gating cell)
Style:                 latch_posedge_precontrol
Module:                top (top)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      clk (/designs/top/ports_in/clk)
                        TCF = (0.50000, 0.333333/ns)
  enable               =      enable (/designs/top/ports_in/enable)
                        TCF = (0.50000, 0.020000/ns)
  test                 =      LOGIC0
Outputs:
  ck_out               =      rc_gclk_148
                        TCF = (0.25000, 0.190000/ns)
Gated FFs:
Module  Clock Gating Instance  Fanout  Gated Flip-flops
-----
top     RC_CG_HIER_INST        8       out1_reg[0]
                                     out1_reg[1]
                                     out1_reg[2]
                                     out1_reg[3]
                                     out1_reg[4]
                                     out1_reg[5]
                                     out1_reg[6]
                                     out1_reg[7]
-----
Total   1                      8
=====
```

1

This report returns

- none for the library cell, indicating that the requested integrated clock-gating cell was not found in the library
- The clock-gating style (latch_posedge_precontrol) which was the requested style, based on the value of the clock-gating directives (attributes)

Low Power in Encounter RTL Compiler

Clock Gating

Reporting Detailed Information

- To list all the clock-gating instances inserted, including the library cell used for the clock-gating cell, the clock-gating style, the signals connected to the inputs and outputs of the gating logic, and the flip-flops gated by this gating cell, enter:

```
report clock_gating -detail
```

The following report gives detailed information for [Example 6-7](#) on page 127. This example has two clock-gating instances: RC_CG_HIER_INST and RC_CG_HIER_INST110.

```
rc:/> report clock_gating -detail
```

```
...
```

```
Detail
```

```
-----
```

```
Clock Gating Instance : RC_CG_HIER_INST
```

```
-----
```

```
Origin:          Inserted by RC
Libcell:         TLATNTSCAX12 (mylib)
Style:           latch_posedge_precontrol
Module:          top (top)
Type:            Leaf level CG Instance
Inputs:
  ck_in          =      clk (/designs/top/ports_in/clk)
                  TCF = (0.50000, 0.333333/ns)
  enable         =      enable (/designs/top/ports_in/enable)
                  TCF = (0.50000, 0.020000/ns)
  test           =      LOGIC0
Outputs:
  ck_out         =      rc_gclk_148
                  TCF = (0.26750, 0.190000/ns)
```

```
Gated FFs:
```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

```
-----
```

top	RC_CG_HIER_INST	8	out1_reg[0]
-----	-----------------	---	-------------

```
....
```

```
-----
```

Total	1	8	
-------	---	---	--

```
=====
```

```
Clock Gating Instance : RC_CG_HIER_INST110
```

```
-----
```

```
Origin:          Inserted by RC
Libcell:         TLATNTSCAX12 (mylib)
Style:           latch_posedge_precontrol
Module:          top (top)
Type:            Leaf level CG Instance
Inputs:
  ck_in          =      n_16 (/designs/top/insta...comb/g117/pins_out/Y)
                  TCF = (0.50000, 0.332500/ns)
  enable         =      enable (/designs/top/ports_in/enable)
                  TCF = (0.50000, 0.020000/ns)
  test           =      LOGIC0
Outputs:
  ck_out         =      rc_gclk_155
                  TCF = (0.27120, 0.195000/ns)
```

```
Gated FFs:
```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

```
-----
```

```
....
```

Clock-Gating Post-Processing

- [Decloning Clock-Gating Instances](#)
- [Removing Clock-Gating Instances](#)

Decloning Clock-Gating Instances

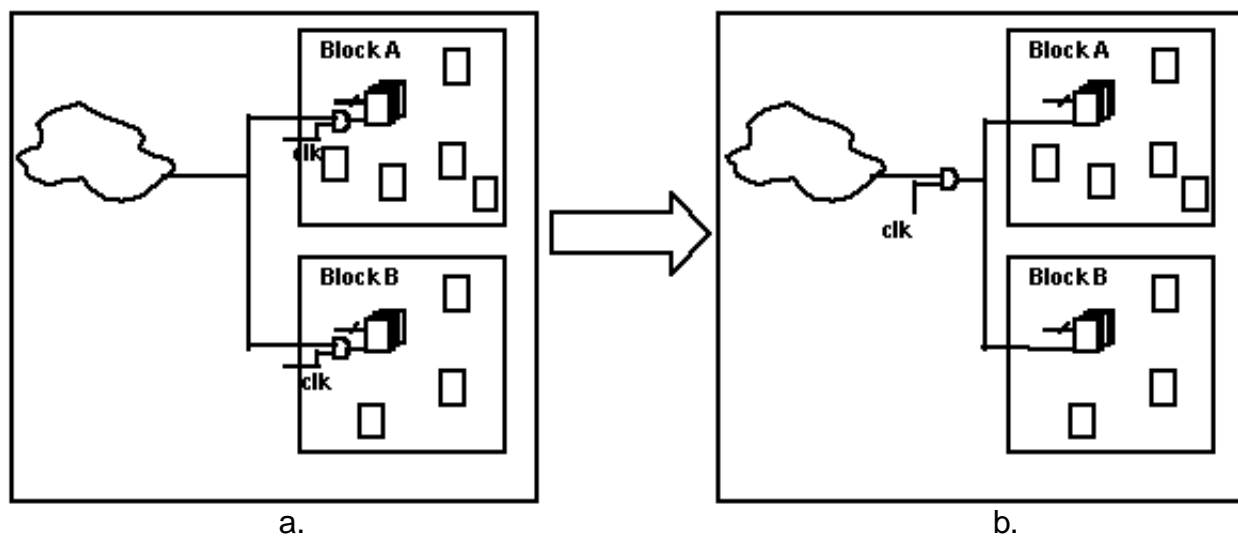
Once clock-gating logic has been inserted, you might decide to merge (declone) clock gating within the same clock domain (enabled by the same set of enable signals) to later create a balanced clock tree and improve the clock skew using a clock tree synthesis tool.

- To merge clock-gating instances, use the following command:

```
clock_gating declone [-hierarchical]
```

This command allows traversing of the design hierarchy using the `-hierarchical` option. This is shown in [Figure 6-24](#) on page 152. [Figure 6-24](#) (a) shows that each module has its own clock-gating logic. [Figure 6-24](#) (b) shows the same design after clock-gating decloning.

Figure 6-24 Merging Clock-Gating Instances



Tip

To merge clock-gating instance candidates, the number of flops they gate together must be less than or equal to the constraint set by the `lp_clock_gating_max_flops` design attribute. Increase the value of the `lp_clock_gating_max_flops` attribute to a large number to enable decloning.

Low Power in Encounter RTL Compiler

Clock Gating

Example 6-10 Decloning Clock-Gating Instances

This example uses a hierarchical design. The `lp_clock_gating_max_flops` attribute setting in [Script 1](#) causes the RC-LP engine to insert six clock-gating instances. [Output 1](#) shows an extract of the detailed clock-gating report. The report shows that the inputs of the first and second, third and fourth, fifth and sixth clock-gating instances are the same, thus indicating that they are candidates for merging.

[Script 2](#) shows the commands used to declone the clock-gating instances. [Output 2](#) shows an extract of the RC output after decloning.

Note: The settings of the `lp_clock_gating_max_flops` attribute are set low in this example for the purpose of showing the concept of decloning. By default, this attribute is set to 2048, to create a minimum of clock-gating instances.

Script 1

```
set_att lp_clock_gating_max_flops 2 /des*/top
synthesize -to_map
report clock_gating -detail
```

Output 1

Detail

Clock Gating Instance : RC_CG_HIER_INST

Origin: Inserted by RC
Libcell: TLATNTSCAX12 (typical)
Style: latch_posedge_precontrol
Module: a (top/u_a)
Type: Leaf level CG Instance
Inputs:
 clk_in = clk (/designs/top/insta..s_comb/g63/pins_out/Y)
 TCF = (0.49380, 0.030000/ns)
 enable = in[1] (/designs/top/ports_in/in1[1])
 TCF = (0.50000, 0.020000/ns)
 test = LOGIC0
Outputs:
 ck_out = rc_gclk_43
 TCF = (0.33380, 0.030000/ns)

Gated FFs:

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

a	RC_CG_HIER_INST	2	out_reg[0] out_reg[1]
---	-----------------	---	--------------------------

Total 1 2

Clock Gating Instance : RC_CG_HIER_INST11

...
Module: a (top/u_a)
Type: Leaf level CG Instance
Inputs:

Low Power in Encounter RTL Compiler

Clock Gating

```

    ck_in      =      clk (/designs/top/insta..s_comb/g63/pins_out/Y)
                    TCF = (0.49380, 0.030000/ns)
    enable     =      in[1] (/designs/top/ports_in/in1[1])
                    TCF = (0.50000, 0.020000/ns)
    test       =      LOGIC0
...
=====

```

Clock Gating Instance : RC_CG_HIER_INST

```

...
Module:          b (top/u_b)
Type:            Leaf level CG Instance
Inputs:
  ck_in          =      n_0 (/designs/top/insta..s_comb/g44/pins_out/Y)
                    TCF = (0.50000, 0.020000/ns)
  enable         =      n_5 (/designs/top/insta..s_comb/g53/pins_out/Y)
                    TCF = (0.50000, 0.020000/ns)
  test           =      LOGIC0
...
=====

```

Clock Gating Instance : RC_CG_HIER_INST41

```

...
Module:          b (top/u_b)
Type:            Leaf level CG Instance
Inputs:
  ck_in          =      n_0 (/designs/top/insta..s_comb/g44/pins_out/Y)
                    TCF = (0.50000, 0.020000/ns)
  enable         =      n_5 (/designs/top/insta..s_comb/g53/pins_out/Y)
  test           =      LOGIC0
...
=====

```

Clock Gating Instance : RC_CG_HIER_INST

```

...
Module:          top (top)
Type:            Leaf level CG Instance
Inputs:
  ck_in          =      tclk (/designs/top/ports_in/tclk)
                    TCF = (0.50000, 0.020000/ns)
  enable         =      n_4 (/designs/top/insta..s_comb/g71/pins_out/Y)
                    TCF = (0.50000, 0.020000/ns)
  test           =      LOGIC0
...
=====

```

Clock Gating Instance : RC_CG_HIER_INST59

```

...
Module:          top (top)
Type:            Leaf level CG Instance
Inputs:
  ck_in          =      tclk (/designs/top/ports_in/tclk)
                    TCF = (0.50000, 0.020000/ns)
  enable         =      n_4 (/designs/top/insta..s_comb/g71/pins_out/Y)
                    TCF = (0.50000, 0.020000/ns)
  test           =      LOGIC0
...
=====

```

Low Power in Encounter RTL Compiler

Clock Gating

Script 2

```
set_att lp_clock_gating_max_flops 4 /des*/top
clock_gating declone -hier
report clock_gating -detail
```

Output 2

```
Setting attribute of design top: 'lp_clock_gating_max_flops' = 4
Decloning clock-gating logic from /
Clock-gating declone status
```

```
=====
Total number of clock-gating instances before: 6
Total number of clock-gating instances after : 3
```

```
...
Clock Gating Instance : RC_CG_DECLONE_HIER_INST
```

```
-----
Origin:                Inserted by RC
Libcell:               TLATNTSCAX12 (typical)
Style:                 latch_posedge_precontrol
Module:                a (top/u_a)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      clk (/designs/top/insta..s_comb/g63/pins_out/Y)
                        TCF = (0.49380, 0.030000/ns)
  enable               =      in[1] (/designs/top/ports_in/in1[1])
                        TCF = (0.50000, 0.020000/ns)
  test                 =      LOGIC0
Outputs:
  ck_out               =      rc_gclk
                        TCF = (0.33380, 0.030000/ns)
```

```
Gated FFs:
```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

a	RC_CG_DECLONE_HIER_INST	4	out_reg[0]
---	-------------------------	---	------------

```
...
out_reg[3]
```

Total	1	4
-------	---	---

```
-----
Clock Gating Instance : RC_CG_DECLONE_HIER_INST
```

```
...
Module:                b (top/u_b)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      n_8 (/designs/top/insta..G_INVERTER/pins_out/Y)
```

Total	1	4
-------	---	---

```
-----
Clock Gating Instance : RC_CG_DECLONE_HIER_INST
```

```
...
Module:                top (top)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      tclk (/designs/top/ports_in/tclk)
```

Total	1	4
-------	---	---

Removing Clock-Gating Instances

The RC-LP engine also provides the capability to remove the inserted clock-gating instances.

- To remove clock-gating instances, use the following command:

```
clock_gating remove [-hierarchical | -cg_list instance_list]
                    [-effort {low|high}]
```

This command allows you to

- Remove all clock-gating instances at the current level of the hierarchy.
- Remove all clock-gating instances in the entire design using the `-hierarchical` option.
- Remove only the specified clock-gating instances using the `-cg_list` option.
- Choose the effort level if performance is a constraint.

Example 6-11 Removing Specific Clock-Gating Instance

Consider [Example 6-7](#) on page 127. The report `clock_gating -detail` command indicates that there are two clock gating instances: `RC_CG_HIER_INST` and `RC_CG_HIER_INST110`.

To remove clock-gating instance `RC_CG_HIER_INST110`, enter:

```
clock_gating remove -cg_list [ find . -inst RC_CG_HIER_INST110]
```

This is similar to

```
clock_gating remove -cg_list /designs/top/instances_hier/RC_CG_HIER_INST110
```

Example 6-12 Removing Clock-Gating Logic in a Hierarchical Design

For this example a hierarchical design is used. The clock-gating report indicates that three clock-gating instances were inserted:

```
=====
...
Module:                top
...
=====

Detail
-----
Clock Gating Instance : RC_CG_HIER_INST
-----
Origin:                Inserted by RC
Libcell:              TLATNTSCAX12 (typical)
Style:                latch_posedge_precontrol
Module:              a (top/u_a)
Type:                Leaf level CG Instance
```

Low Power in Encounter RTL Compiler

Clock Gating

```

Inputs:
  ck_in      =      clk (/designs/top/insta..s_comb/g60/pins_out/Y)
               TCF = (0.42120, 0.030000/ns)
  enable     =      in[1] (/designs/top/ports_in/in1[1])
               TCF = (0.50000, 0.020000/ns)
  test       =      RC_CG_TEST_PORT (/designs/top/insta../pins_out/scan_enable)
               TCF = (0.27750, 0.020000/ns)

```

```

Outputs:
  ck_out     =      rc_gclk_44
               TCF = (0.24250, 0.025000/ns)

```

Gated FFs:

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

a	RC_CG_HIER_INST	4	out_reg[0]
---	-----------------	---	------------

...

Total	1	4	
-------	---	---	--

Clock Gating Instance : RC_CG_HIER_INST

```

...
Module:      b (top/u_b)
Inputs:
  ck_in      =      n_0 (/designs/top/insta..s_comb/g41/pins_out/Y)
               TCF = (0.50120, 0.020000/ns)
  enable     =      n_4 (/designs/top/insta..s_comb/g50/pins_out/Y)
               TCF = (0.50000, 0.020000/ns)
  test       =      RC_CG_TEST_PORT (/designs/top/insta../pins_out/scan_enable)
               TCF = (0.27750, 0.020000/ns)

```

```

Outputs:
  ck_out     =      rc_gclk_71
               TCF = (0.44120, 0.020000/ns)

```

Gated FFs:

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

b	RC_CG_HIER_INST	4	out_reg[0]
---	-----------------	---	------------

...

Total	1	4	
-------	---	---	--

Clock Gating Instance : RC_CG_HIER_INST

```

...
Module:      top (top)
Type:      Leaf level CG Instance
Inputs:
  ck_in      =      tclk (/designs/top/ports_in/tclk)
               TCF = (0.50000, 0.020000/ns)
  enable     =      n_2 (/designs/top/insta..s_comb/g69/pins_out/Y)
               TCF = (0.50000, 0.020000/ns)
  test       =      n_8 (/designs/top/insta../pins_out/scan_enable)
               TCF = (0.27750, 0.020000/ns)

```

Outputs:

...

Gated FFs:

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

top	RC_CG_HIER_INST	4	out3_reg[0]
-----	-----------------	---	-------------

...

Low Power in Encounter RTL Compiler

Clock Gating

```
-----
Total      1                                4
=====
```

An inspection of the design hierarchy shows:

```
rc:/designs/top> ls instances_hier
/designs/top/instances_hier:
./                RC_CG_HIER_INST/      u_a/                u_b/                u_jtag/
rc:/designs/top> cd u_a/instances_hier
rc:/designs/top/instances_hier/u_a/instances_hier> ls
./                RC_CG_HIER_INST/
rc:/designs/top/instances_hier/u_a/instances_hier> cd ../../u_b/instances_hier
rc:/designs/top/instances_hier/u_b/instances_hier> ls
./                RC_CG_HIER_INST/
```

To remove the clock-gating instance from instance `u_a`, either

- Enter the following command from root:

```
rc:/> clock_gating remove -cg_list u_a/*/RC_CG_HIER_INST
Removing clock-gating logic from /
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'u_a/RC_CG_HIER_INST'.
```

- Enter the following command if you navigated down to instance `u_a`:

```
rc:/designs/top/instances_hier/u_a> clock_gating remove -cg_list \
[ find . -inst RC_CG_HIER_INST]
Removing clock-gating logic from /designs/top/instances_hier/u_a
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'u_a/RC_CG_HIER_INST'.
```

To remove all `clock_gating` -instances, you can enter one of the following commands:

- `rc:/> clock_gating remove -hier`

```
rc:/> clock_gating remove -hier
Removing clock-gating logic from /
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'u_a/RC_CG_HIER_INST'.
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'u_b/RC_CG_HIER_INST'.
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'RC_CG_HIER_INST'.
```

- `rc:/designs/top> clock_gating remove -cg_list [find . -inst RC_CG_HIER_INST]`

```
rc:/designs/top> clock_gating remove -cg_list [ find . -inst RC_CG_HIER_INST]
Removing clock-gating logic from /designs/top
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'u_a/RC_CG_HIER_INST'.
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'u_b/RC_CG_HIER_INST'.
Info      : Removed a clock-gating instance. [CG-400]
           : Removed clock-gating instance 'RC_CG_HIER_INST'.
```

Note: This command happens to remove all clock gating instances from the design because they all have the same name in this example.

Multi-Stage Clock Gating

Using regular clock-gating insertion, the clock-gating instances each are driven by one *resultant* enable signal. Multi-staging allows you to have more granular control over the enable control logic. The RC-LP engine provides the following capabilities:

- Creating Shared Clock-Gating Logic Using Common Enable
- Splitting Clock-Gating Instances into Multiple Levels of Clock-Gating Logic
- Consolidating Multi-Stage Clock-Gating Logic

Creating Shared Clock-Gating Logic Using Common Enable

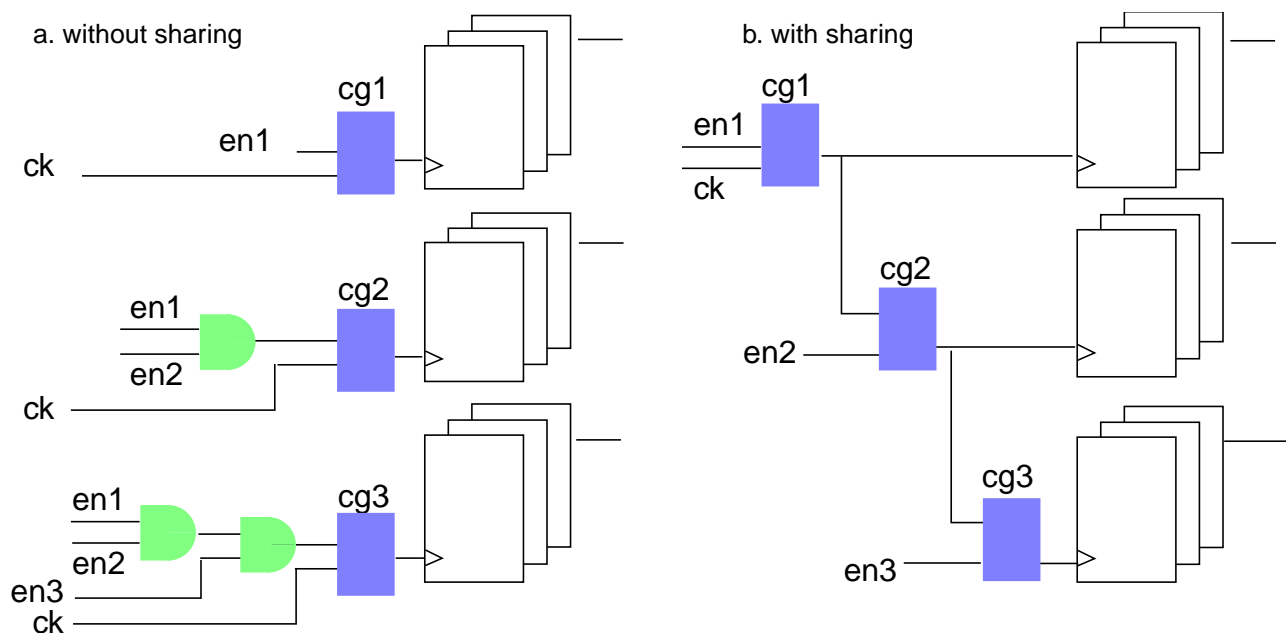
By extracting the common enable sub-function of leaf level clock-gating instances, the RC-LP engine can create multi-stage clock-gating logic. This feature shown in Figure 6-25, helps to reduce the switching activities on the clock tree. Additionally, by applying a simplified enable logic to each clock-gating instance, timing closure can be achieved more easily.

- To create shared clock-gating logic using a common enable function, use this command:

```
clock_gating_share [-hierarchical] [-max_level integer]
```

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Figure 6-25 Clock Gating with and Without Sharing



Example 6-13 Creating Shared Clock-Gating Logic Using Common Enable

This example contains three enable signals. The two sets of registers are each controlled by two enable signals. Both sets share the inverted `en1` signal (see [RTL Code](#)).

[Script](#) list the commands needed for regular clock-gating insertion and the commands to extract the enable function shared by the clock-gating instances and to insert shared clock-gating logic with the common enable subfunction as the enable signal.

[Clock-Gating Report before Sharing](#) shows two clock-gating instances were inserted.

- The `Type info Leaf level CG Instances` indicates that both clock-gating instances are driving flip-flops.
- The enable signals of the clock-gating instances are the output of combinational logic.

[Output after Sharing](#) shows

- The messages issued by the `clock_gating share` command
The messages indicate that one shared clock-gating instance was added.
- The output of the `report clock_gating -multi_stage` command
The output shows the multi-stage hierarchy of the clock-gating logic. Instance `RC_CG_SHARED_HIER_L1_INST` is driving the two leaf clock-gating instances (`RC_CG_HIER_INST` and `RC_CG_HIER_INST19`).
- A detailed clock-gating report
The `Type info Stage 1 CG Instance` indicates a multi-stage clock-gating instance driving one or more Leaf level CG Instance(s).
The enable signals of the original clock-gating instances have changed.
The clock output of `RC_CG_SHARED_HIER_L1_INST` is the clock input for the two clock-gating instances that it is driving.

RTL Code

```
module test(clk, en1, en2, en3, a, b, out1, out2);
input clk, en1, en2, en3;
input [3:0] a, b;
output [3:0] out1, out2;
reg [3:0] out1, out2;
always @ (posedge clk)
begin
    if (!en1 && en2)
        out1 <= a;
    if (!en1 && en3)
        out2 <= b;
end
endmodule
```


Low Power in Encounter RTL Compiler

Clock Gating

Script

```
set_attr library my.lib
set_attr lp_insert_clock_gating true /
read_hdl basic.v
elab
define_clock -n clk -p 10 [clock_ports]
synth -to_map
report clock_gating -detail
clock_gating_share -hier
report clock_gating -multi_stage
report clock_gating -detail
```

Clock-Gating Report before Sharing

```
...
Clock Gating Instance : RC_CG_HIER_INST
-----
```

```
...
Module:                test (test)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      clk (/designs/test/ports_in/clk)
                        TCF = (0.50000, 200.000000/ns)
  enable               =      n_5 (/designs/test/inst..s_comb/g25/pins_out/Y)
                        TCF = (0.22120, 0.025000/ns)
  test                 =      TEST_MODE (/designs/test/ports_in/TEST_MODE)
                        TCF = (0.50000, 0.020000/ns)
Outputs:
  ck_out               =      rc_gclk_51
                        TCF = (0.32120, 1.217500/ns)
Gated FFs:
```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

test	RC_CG_HIER_INST	4	out1_reg[0]
...			

```
-----
Clock Gating Instance : RC_CG_HIER_INST19
-----
```

```
...
Module:                test (test)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =      clk (/designs/test/ports_in/clk)
                        TCF = (0.50000, 200.000000/ns)
  enable               =      n_6 (/designs/test/inst..s_comb/g24/pins_out/Y)
                        TCF = (0.28380, 0.020000/ns)
  test                 =      TEST_MODE (/designs/test/ports_in/TEST_MODE)
                        TCF = (0.50000, 0.020000/ns)
Outputs:
  ck_out               =      rc_gclk_56
                        TCF = (0.34750, 1.325000/ns)
Gated FFs:
```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

test	RC_CG_HIER_INST19	4	out2_reg[0]
...			

Low Power in Encounter RTL Compiler

Clock Gating

The enable signal `n_5` of the first clock-gating instance is the output of combinational logic. The following commands trace the inputs of this logic:

```
rc:/> get_att driver [find / -net n_5]
/designs/test/instances_comb/g25/pins_out/Y
rc:/> ls /designs/test/instances_comb/g25/pins_in
/designs/test/instances_comb/g25/pins_in:
./
      AN      B
rc:/> get_att net /designs/test/instances_comb/g25/pins_in/AN
/designs/test/nets/en2
rc:/> get_att net /designs/test/instances_comb/g25/pins_in/B
/designs/test/nets/en1
```

Similar commands show that the enable signal for the second clock-gating instance is a combination of enable signals `en1` and `en3`.

Output after Sharing

```
rc:/> clock_gating share -hier
clock_gating share: inserts shared clock-gating logic for enable logic shared
between already inserted clock-gating logic
Inserting shared clock-gating logic from '/designs/test'...
Inserted 1 shared clock-gating instance.
```

```
0
rc:/> report clock_gating -multi_stage
...
```

CG Instance	Level	Fanouts
RC_CG_SHARED_HIER_L1_INST	1	2
RC_CG_HIER_INST	0	4
RC_CG_HIER_INST19	0	4

```
rc:/> report clock_gating -det
...
```

Detail

Clock Gating Instance : RC_CG_HIER_INST

```
...
Type:                      Leaf level CG Instance
Inputs:
  ck_in      =      rc_gclk (/designs/test/inst..._INST/pins_out/ck_out)
               TCF = (0.39880, 1.497500/ns)
  enable     =      en2 (/designs/test/ports_in/en2)
               TCF = (0.50000, 0.020000/ns)
  test       =      TEST_MODE (/designs/test/ports_in/TEST_MODE)
               TCF = (0.50000, 0.020000/ns)
Outputs:
  ck_out     =      rc_gclk_51
               TCF = (0.32120, 1.217500/ns)
Gated FFs:
```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
--------	-----------------------	--------	------------------

...

Clock Gating Instance : RC_CG_HIER_INST19

Low Power in Encounter RTL Compiler

Clock Gating

```

-----
...
Type:                Leaf level CG Instance
Inputs:
  ck_in              =    rc_gclk (/designs/test/inst.._INST/pins_out/ck_out)
                        TCF = (0.39880, 1.497500/ns)
  enable             =    en3 (/designs/test/ports_in/en3)
                        TCF = (0.50000, 0.020000/ns)
  test               =    TEST_MODE (/designs/test/ports_in/TEST_MODE)
                        TCF = (0.50000, 0.020000/ns)
Outputs:
  ck_out             =    rc_gclk_56
                        TCF = (0.34750, 1.325000/ns)
Gated FFs:
Module  Clock Gating Instance  Fanout  Gated Flip-flops
-----
...

```

Clock Gating Instance : **RC_CG_SHARED_HIER_L1_INST**

```

-----
...
Module:              test (test)
Type:                Stage 1 CG Instance
Inputs:
  ck_in              =    clk (/designs/test/ports_in/clk)
                        TCF = (0.50000, 200.000000/ns)
  enable             =    n_9 (/designs/test/inst..G_INVERTER/pins_out/Y)
                        TCF = (0.50000, 0.020000/ns)
  test               =    TEST_MODE (/designs/test/ports_in/TEST_MODE)
                        TCF = (0.50000, 0.020000/ns)
Outputs:
  ck_out             =    rc_gclk
                        TCF = (0.39880, 1.497500/ns)
-----

```

3

The two leaf clock-gating instances are now controlled by a single enable signal, while the enable signal of the shared clock-gating instance (RC_CG_SHARED_HIER_L1_INST) is driven by net n_9. The following commands determine that net n_9 is the inverted en1 signal, which was the common enable signal.

```

rc:/> get_att driver [find / -net n_9]
/designs/test/instances_comb/RC_CG_INVERTER/pins_out/Y

rc:/> get_att net /designs/test/instances_comb/RC_CG_INVERTER/pins_in/*
/designs/test/nets/en1

```

Splitting Clock-Gating Instances into Multiple Levels of Clock-Gating Logic

Even if sharing a common enable function is not possible, the RC-LP engine can transform a single stage of clock gating into multi-stage clock gating (see [Figure 6-26](#) on page 164) based on timing or power considerations. You can use this feature to help achieve timing closure when the leaf enable signal is on the critical path for timing. By splitting into multi stages, the RC-LP engine can apply a simplified enable logic to each clock-gating instance, and as a result, the RC-LP engine can move the faster enable signal closer to the inputs.

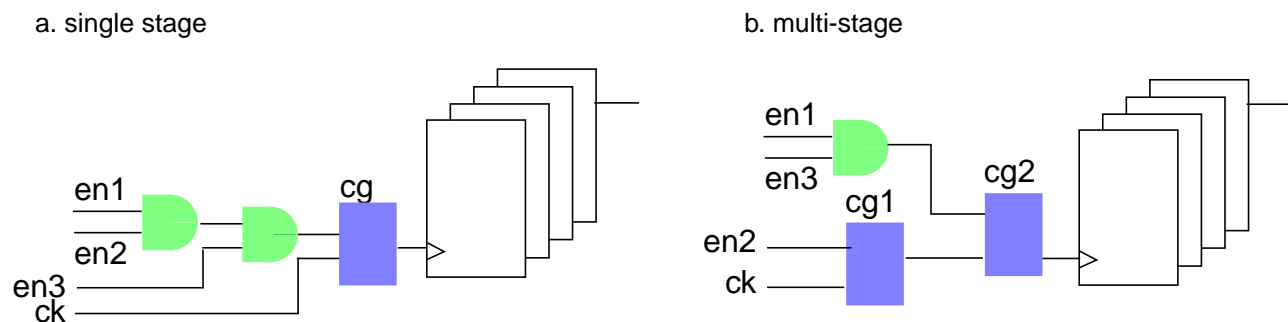
This feature can also improve clock tree power savings if the leaf level clock-gating instance has a large fanout because the switching activities on the clock subtree are reduced.

- To transform a single stage of clock gating into multi-stage clock gating, use the following command:

```
clock_gating_split [-hierarchical] [-max_level integer]  
                  [-power_driven] [-start_from instance]
```

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Figure 6-26 Clock Gating before and after Splitting



Example 6-14 Transform a Single Stage of Clock Gating into Multi-Stage Clock Gating

This example contains one set of registers controlled by three enable signals (see [RTL Code](#)). [Script](#) shows the commands used for the regular clock-gating insertion and splitting into multiple stages.

[Clock-Gating Report before Splitting](#) shows that after regular clock-gating insertion only one clock-gating instance was inserted. The enable signal of this clock-gating instance is the output of combinational logic.

[Output after Splitting](#) shows

- The messages issued by the `clock_gating split` command

Because the design has three enable signals, the enable function can be split two times. This is achieved by setting the value of the `-max_level` option to 2. The messages also indicate two rounds of splitting.

- The output of the `report clock_gating -multi_stage` command

The output shows the multi-stage hierarchy of the clock-gating logic. Instance `RC_CG_SPLITTED_HIER_INST` drives instance `RC_CG_SPLITTED_HIER_INST111`, which in turn drives the original clock-gating instance.

- A detailed clock-gating report

The `Type` info in the detailed report indicates the position of the clock-gating instance:

- *Leaf level CG Instance* is a clock-gating instance driving the flip-flops.
- *Stage 1 CG Instance* is a multi-stage clock-gating instance driving the *Leaf level CG Instance*.
- *Stage 2 CG Instance* is a multi-stage clock-gating instance driving the *Stage 1 CG Instance*.

Each stage is driven by a simple enable signal.

RTL Code

```
module cnt_mul ( a, b, c, clk, e1, e2, e3, cnt);
input clk, e1, e2, e3;
input [4:0] a, b, c;
output [9:0] cnt;

reg [9:0] cnt;

always @(posedge clk)
begin
    if (e1 & e2)
        if (e2 & e3)
            if(e3 & e1)
```

Low Power in Encounter RTL Compiler

Clock Gating

```

        cnt = a + b + c;
    end
endmodule

```

Script

```

set_attr library my.lib
set_attr lp_insert_clock_gating true
read_hdl mydesign.v
elab
set_attr lp_clock_gating_min_flops 1 des*/*

synthesize -to_map
report clock_gating -det

clock_gating split -max_level 2
report clock_gating -multi_stage
report clock_gating -det

```

Clock-Gating Report before Splitting

Detail

```

-----
Clock Gating Instance : RC_CG_HIER_INST
-----

...
Style:                latch_posedge_precontrol
Module:               cnt_mul (cnt_mul)
Type:                 Leaf level CG Instance
Inputs:
  ck_in               =      clk (/designs/cnt_mul/ports_in/clk)
                        TCF = (0.50000, 0.020000/ns)
  enable              =      n_19 (/designs/cnt_mul/i..._comb/g104/pins_out/Y)
                        TCF = (0.18500, 0.020000/ns)
  test                =      LOGIC0
Outputs:
  ck_out              =      rc_gclk
                        TCF = (0.00000, 0.000000/ns)

Gated FFs:
Module  Clock Gating Instance  Fanout  Gated Flip-flops
-----
cnt_mul  RC_CG_HIER_INST        7       cnt_reg[0]
                                     cnt_reg[1]
                                     ...
                                     cnt_reg[6]
-----

```

1

The enable signal n_19 of the clock-gating instance is the output of combinational logic. The following commands trace the inputs of this logic:

```

rc:/> get_att driver [find / -net n_19]
/designs/cnt_mul/instances_comb/g104/pins_out/Y
rc:/> ls    /designs/cnt_mul/instances_comb/g104/pins_in
/designs/cnt_mul/instances_comb/g104/pins_in:
./          A          B          C

```

Low Power in Encounter RTL Compiler

Clock Gating

```
rc:/> get_att net /designs/cnt_mul/instances_comb/g104/pins_in/A
/designs/cnt_mul/nets/e1
rc:/> get_att net /designs/cnt_mul/instances_comb/g104/pins_in/B
/designs/cnt_mul/nets/e2
rc:/> get_att net /designs/cnt_mul/instances_comb/g104/pins_in/C
/designs/cnt_mul/nets/e3
```

Output after Splitting

```
rc:/> clock_gating split -max_level 2
```

Clock-gating Split Status (round 1)

Category	Number	Percentage
Clock-gating splitted	1	100%
Clock-gating not splitted		
Enable non-splittable	0	0%
Enable with multiple fanouts	0	0%
Non-combinational enable	0	0%
Preserved	0	0%
Clock-gating before splitting	1	100%
Clock-gating after splitting	2	

Clock-gating Split Status (round 2)

Category	Number	Percentage
Clock-gating splitted	1	50%
Clock-gating not splitted		
Enable non-splittable	0	0%
Enable with multiple fanouts	0	0%
Non-combinational enable	1	50%
Preserved	0	0%
Clock-gating before splitting	2	100%
Clock-gating after splitting	3	

```
rc:/> report clock_gating -multi_stage
```

...

CG Instance	Level	Fanouts
RC_CG_SPLITTED_HIER_INST	2	1
RC_CG_SPLITTED_HIER_INST111	1	1
RC_CG_HIER_INST	0	7

```
rc:/> report clock_gating -detail
```

...

Detail

Clock Gating Instance : RC_CG_HIER_INST

```
-----
Origin:          Inserted by RC
Libcell:         none (RC-LP created clock-gating logic)
Style:           latch_posedge_precontrol
Module:          cnt_mul (cnt_mul)
Type:            Leaf level CG Instance
```

Low Power in Encounter RTL Compiler

Clock Gating

```

Inputs:
  ck_in      =      n_26 (/designs/cnt_mul/i..ST106/pins_out/ck_out)
              TCF = (0.000000, 0.000000/ns)
  enable     =      e3 (/designs/cnt_mul/ports_in/e3)
              TCF = (0.500000, 0.020000/ns)
  test       =      LOGIC0
Outputs:
  ck_out     =      rc_gclk
              TCF = (0.000000, 0.000000/ns)

```

Module	Clock Gating Instance	Fanout	Gated Flip-flops
cnt_mul	RC_CG_HIER_INST	7	cnt_reg[0]
			...

Clock Gating Instance : **RC_CG_SPLITTED_HIER_INST**

```

Origin:      Inserted by RC
Libcell:     none (RC-LP created clock-gating logic)
Style:       latch_posedge_precontrol
Module:      cnt_mul (cnt_mul)
Type:        Stage 2 CG Instance
Inputs:
  ck_in      =      clk (/designs/cnt_mul/ports_in/clk)
              TCF = (0.500000, 0.020000/ns)
  enable     =      e1 (/designs/cnt_mul/ports_in/e1)
              TCF = (0.500000, 0.020000/ns)
  test       =      LOGIC0
Outputs:
  ck_out     =      n_23
              TCF = (0.05120, 0.010000/ns)

```

Clock Gating Instance : **RC_CG_SPLITTED_HIER_INST106**

```

Origin:      Inserted by RC
Libcell:     none (RC-LP created clock-gating logic)
Style:       latch_posedge_precontrol
Module:      cnt_mul (cnt_mul)
Type:        Stage 1 CG Instance
Inputs:
  ck_in      =      n_23 (/designs/cnt_mul/i.._INST/pins_out/ck_out)
              TCF = (0.05120, 0.010000/ns)
  enable     =      e2 (/designs/cnt_mul/ports_in/e2)
              TCF = (0.500000, 0.020000/ns)
  test       =      LOGIC0
Outputs:
  ck_out     =      n_26
              TCF = (0.000000, 0.000000/ns)

```


Consolidating Multi-Stage Clock-Gating Logic

The RC-LP engine can consolidate multi-stage clock-gating logic into a single stage or fewer clock-gating instance(s). You can use this feature to reduce area if the design is not timing-critical. By joining multiple stages, the RC-LP engine can reduce the number of clock-gating instances.

- To consolidate multi-stage clock gating instances into a single or fewer clock-gating instance(s), use the following command:

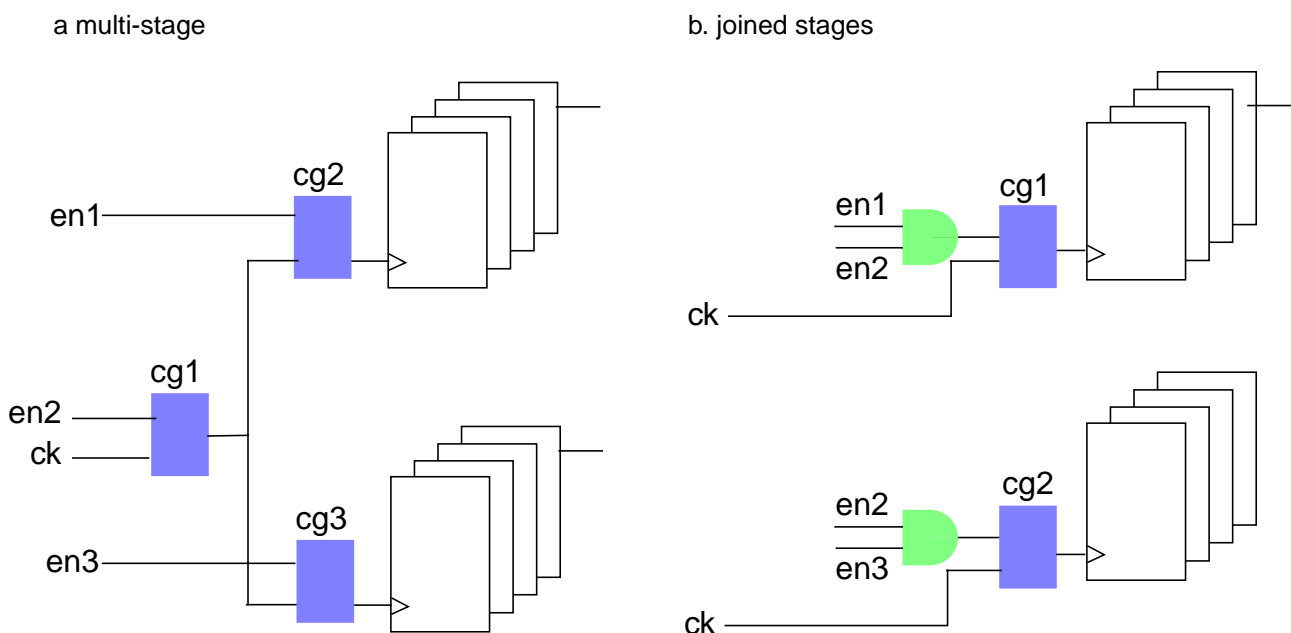
```
clock_gating_join [-hierarchical] [-max_level integer]  
                  [-multi_fanouts] [-start_from instance]
```

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

In Figure 6-27, the root clock-gating instance *cg1* is driving multiple clock-gating instances (*cg2* and *cg3*). By default, the `clock_gating_join` command would not have any impact in this case. If, however, you specify the `clock_gating_join` command with the `-multi_fanouts` option, you allow consolidation as is shown in Figure 6-27 (b). Other options give you further control over how many stages you consolidate and where in the design you want to do this operation.

Note: Consolidation of clock-gating instances starts at the root clock-gating instance.

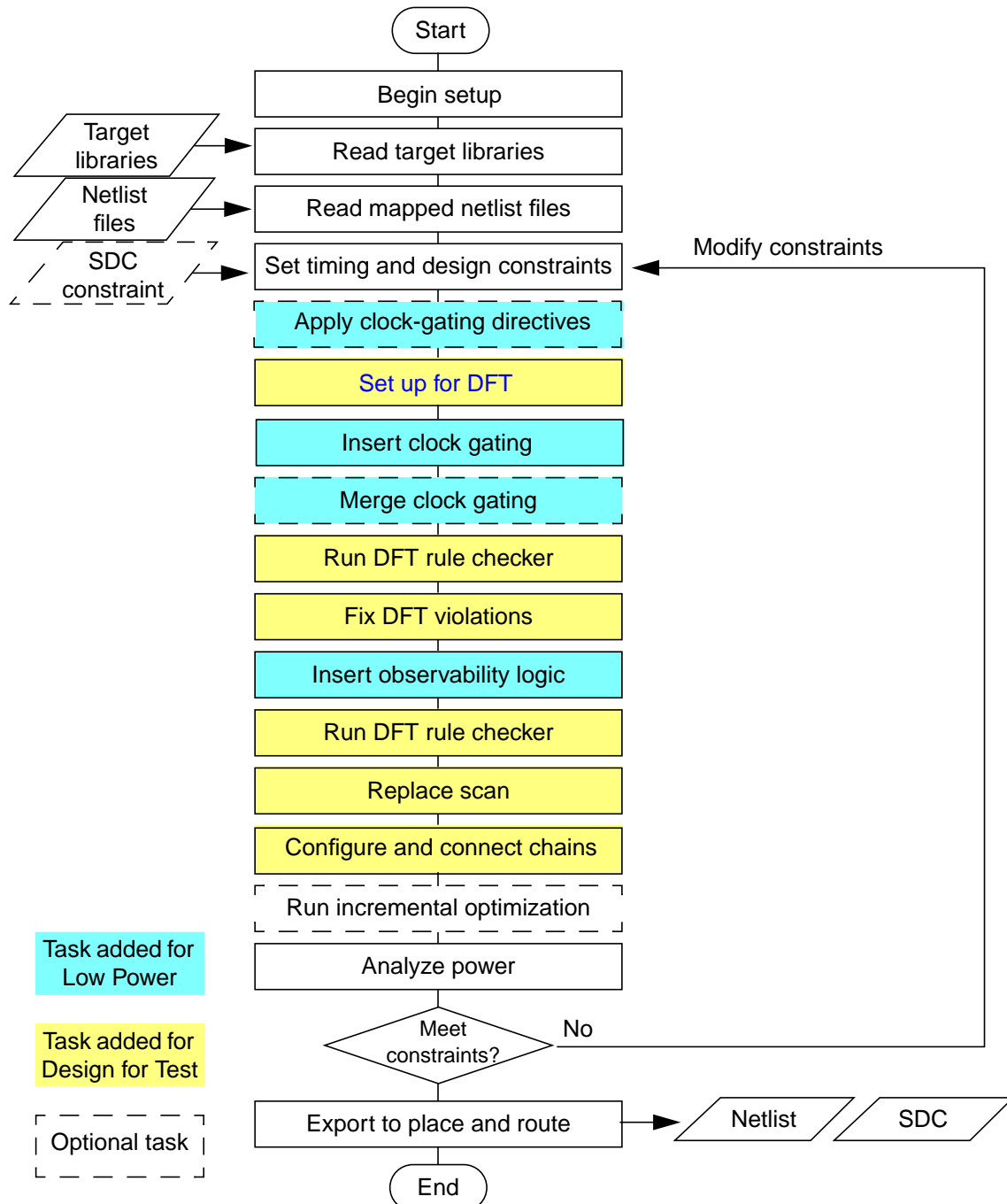
Figure 6-27 Clock Gating before and after Consolidation



Inserting Clock Gating in a Mapped Netlist

This section briefly describes how to insert clock gating in a mapped netlist. Figure 6-28 shows the additional steps in the low power flow.

Figure 6-28 Clock-Gating Flow with DFT Starting from Mapped Netlist



Low Power in Encounter RTL Compiler

Clock Gating

This approach applies when you performed synthesis with a third-party tool and you want to use RTL Compiler to insert clock gating in the netlist generated by this third-party tool, and (optionally) optimize the netlist further.

Note: The RC-LP engine does not recognize clock-gating logic inserted by a third-party tool.

When starting with a structural netlist, perform the following steps to insert clock gating:

1. General setup.

```
set_attribute lib_search_path ...
set_attribute hdl_search_path ...
```

2. Read the target libraries.

```
set_attribute library library_list /
```

3. Read the structural netlist.

```
read_hdl -structural mapped_netlist
```

4. (Optional) Set timing and design constraints (including power constraints).

5. (Optional) Specify attributes to control handling of timing exceptions during clock gating, select clock-gating logic, and to control insertion of clock-gating logic.

6. (If you want clock gating with DFT) Specify the DFT setup.

```
define_dft shift_enable -name se_signal -active high se_port...
define_dft test_mode -name tpi -active high test_mode_port...
```

(If you want clock gating with DFT) Identify the test mode signal used for the clock-gating logic to the RC-LP engine.

```
set_att lp_clock_gating_test_signal /des*/design/dft/test_signals/tpi
/des*/design
```

Note: The clock-gating test signal can be either a shift enable or a test mode signal, and must have been defined already with either the `define_dft test_mode` or the `define_dft shift_enable` command.

7. Insert clock gating.

```
clock_gating_insert_in_netlist
report clock_gating
```

8. (Optional) Merge clock-gating instances.

```
clock_gating_declone [-hierarchical]
```

9. (If you want clock gating with DFT) Run the DFT rule checker.

```
check_dft_rules
report dft_registers
```

Low Power in Encounter RTL Compiler

Clock Gating

- 10.** (If you want clock gating with DFT) Fix DFT rule violations.

```
fix_dft_violations...
```

- 11.** (If you want clock gating with DFT) Insert the observability logic of the clock-gating instances.

```
clock_gating_insert_obs...
```

- 12.** (If you want clock gating with DFT) Run the DFT rule checker, to determine the DFT status of the observability logic.

```
check_dft_rules
```

- 13.** (If you want clock gating with DFT) Convert the flip-flops which pass the DFT rule checker to their scan flip-flops.

```
replace_scan
```

- 14.** (If you want clock gating with DFT) Configure and connect the scan chains.

```
connect_scan_chains -auto_create_chains
```

- 15.** To fix any timing issues (including timing issues introduced by inserting the observability logic and by connecting the scan chains), run incremental optimization:

```
synthesize -incremental
```

- 16.** Export the design.

```
write_hdl > ml.vm  
...
```

Troubleshooting

Clock-Gating Decloning is Not Happening

Check if any of a clock-gating instance pins or nets connected to its pins are marked preserved. In that case, the clock-gating instance is not considered for decloning.

Not Expected Power Savings after Clock Gating

Clock gating is a technique commonly used to reduce the dynamic power consumption of a design. Often, clock gating also results in a reduction of chip area, because enabled flops are replaced with simplified flops after clock-gating insertion.

If the enable signal does not switch very often and stays active most of the time, you might not see any considerable power savings by adding clock gating to those registers. You can check this using the following commands:

► `rc:/designs/design> report power enable_net`

Check the switching activity of the enable signal. If the probability is close to one and the toggle rate is high, no considerable power savings should be expected from clock gating.

If the registers are not enabled that often and the power savings after clock-gating insertion are not trivial, check whether the major power consumption comes from macros such as memory and IO pads. You can use the following commands:

■ `rc:/designs/design> report gates -power`

Check the percentage of power consumed by the `timing_model` type. Memories and IO pads would belong to this category. If a large percentage of the power is consumed by this category, clock gating will not have much effect on the power consumption.

If a large percentage of the power is consumed by sequential cells, clock gating can have an effect. In this case, check the percentage of gated flip-flops. You can then conclude whether more power savings can be achieved.

■ `rc:/designs/design> report power -nworst 10`

Lists the ten top instances that consume the most power. Again the type of instance that consumes the most power can help you conclude whether more power savings can be achieved with clock gating.

Limitations

- If you have multi-stage clock-gating logic inserted, the RC-LP engine can only remove the leaf level clock-gating instances.
- The clock-gating insertion in a mapped netlist currently does not handle complex MUXes and flip-flops with a synchronous reset. The insertion is limited to netlists with flip-flops that have a Q-to-D feedback loop through a two-input MUX.

Note: For synchronous enable flip-flops, the RC-LP engine will insert clock-gating logic and remap the synchronous enable flip-flop to a DFF if it is available in the technology library. Otherwise, it will tie the enable pin of the flip-flop to constant 1'b1 or 1'b0 depending on the phase of the synchronous enable.

- The RC-LP engine cannot remove clock-gating logic that is gating scan flops when the scan flops are part of a scan chain.
- The low-effort mode for clock gating removal has the following limitations:
 - Flops that have no unique data pin, will not be ungated.
However, such flops can be ungated using the high-effort mode.
 - When a clock gating instance is removed, the RC-LP engine adds the MUX back and the reconstructs the feedback loop from the Flip-flop to the MUX. This results in an area penalty.

Note: In high-effort mode, the feedback logic for the ungated flops is optimized.

Operand Isolation

- [Introduction](#) on page 176
- [RTL Coding Style Examples](#) on page 179
- Tasks
 - [Enabling Operand Isolation](#) on page 180
 - [Controlling Naming of Modules, Nets, and Ports](#) on page 180
 - [Finding Operand-Isolation Logic in the Design Hierarchy](#) on page 180
 - [Checking System Messages during Optimization](#) on page 181
 - [Reporting Operand Isolation Information](#) on page 182
- [Limitations](#) on page 185

Introduction

Although clock gating is very effective in reducing the dynamic power dissipation of a digital circuit, it can only save power on sequential elements and clock circuitry. Operand Isolation is a dynamic power optimization technique that can reduce power dissipation in datapath blocks controlled by an enable signal.

An example of restricted power savings is shown in Figure 7-1. In the digital system shown in Figure 7-1 (a), register C uses the result of the multiplier when the `Enable` is on. When the `Enable` is off, register C only uses the result of register B, but the multiplier continues its computations. Because the multiplier dissipates the most power, the total amount of power wasted is quite significant.

Clock gating cannot be used in this case, because the output of register B is always used by either the multiplier or the comparator. One solution to this problem is to shut down (isolate) the function unit (operand) when its results are not used. As shown in Figure 7-1 (b), the RC-LP engine inserts AND gates at the inputs of the multiplier and uses the enable logic of the multiplier to gate the signal transitions. As a result no dynamic power is dissipated when the result of the multiplier is not needed.

Figure 7-1 Operand Isolation Concept

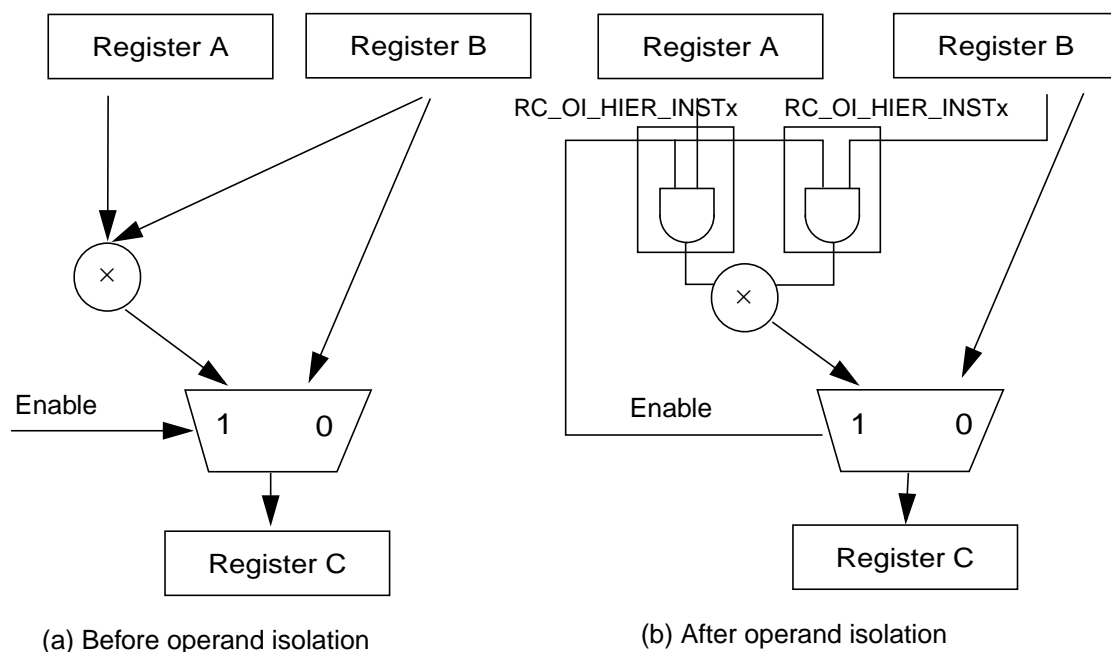
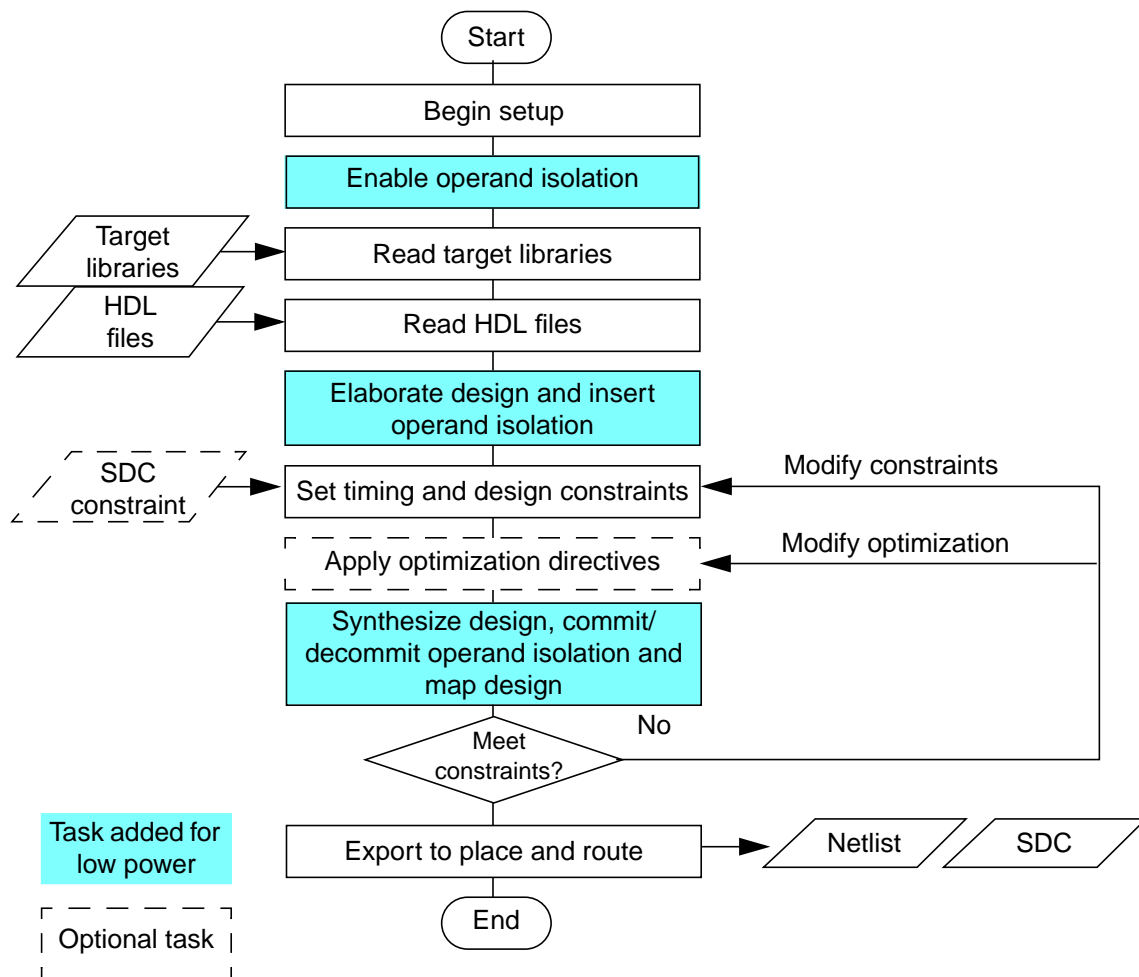


Figure 7-2 on page 177 highlights the tasks you need to add to the generic synthesis flow to use the operand-isolation flow starting from RTL.

Low Power in Encounter RTL Compiler

Operand Isolation

Figure 7-2 Recommended Operand Isolation Flow Starting from RTL



Tasks for the Recommended Operand Isolation Flow Starting from RTL

- Enabling Operand Isolation before elaboration
- During elaboration, given the RTL netlist of the design, the RC-LP engine identifies the datapath block candidates (such as adders and multipliers) for operand isolation and inserts operand isolation instances.
Note: Only datapath blocks with an input width equal to or larger than 8 are considered.
- Commitment and decommitment of the operand isolation instances during synthesis

Low Power in Encounter RTL Compiler

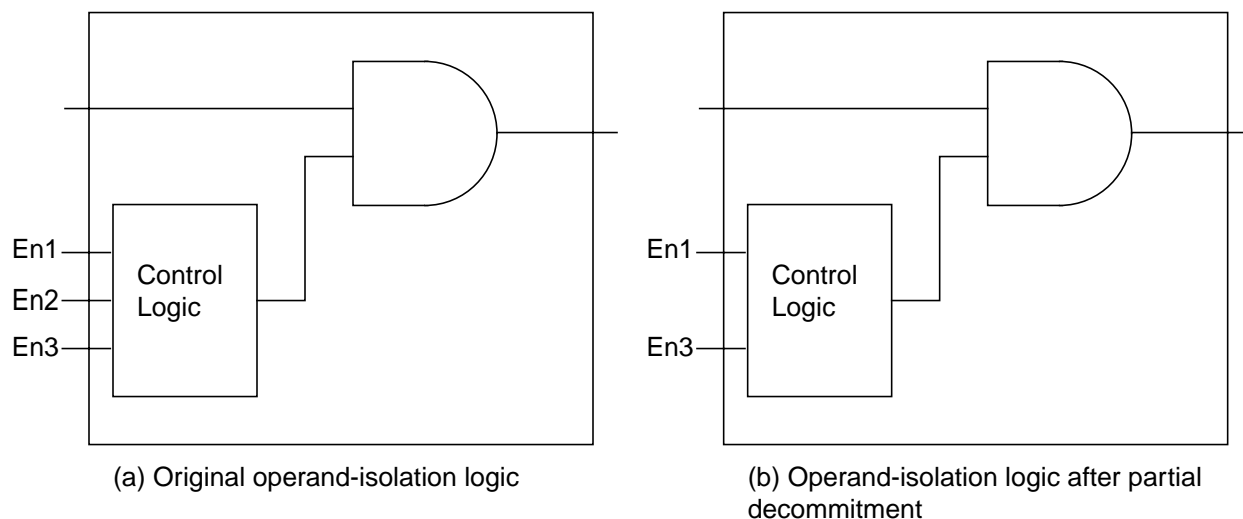
Operand Isolation

Commitment and decommitment are based on timing and power considerations. Some criteria used are:

- ❑ If the enable used for the operand isolation instance is on a timing-critical path, the operand isolation instance is decommitted.
- ❑ If an input of a datapath block is on a timing-critical path, all operand isolation instances on that path are decommitted.
- ❑ If adding an operand isolation instance increases power, the operand isolation instance is decommitted.
- ❑ If the input of a datapath block is used multiple times, only one operand isolation instance is considered for this input.
- ❑ If two or more enable signals are used for operand isolation logic, and one of the enable signals has a large delay, this enable signal will not be used for the operand isolation logic. This is called *partial* decommitment.

In Figure 7-3 (a), enable signal $En2$ is part of the original operand-isolation logic. If during synthesis the RC-LP engine identifies that this enable signal is on a critical path, and removing this signal from the operand isolation logic does not change the functionality of the circuit, the RC-LP engine will remove the signal from the OI logic as shown in Figure 7-3(b).

Figure 7-3 Partial Decommitment



■ Reporting Operand Isolation Information after synthesis

RTL Coding Style Examples

The following two examples show two different RTL coding styles. Example 7-1 uses nested enable statements. In this case, the RC-LP engine determines that the condition under which the results of the adder are useful is $(en[1] \ \& \ en[0])$ and the condition under which the results of the subtractor are useful is also $(en[1] \ \& \ en[0])$. The RC-LP engine stores the logic in the operand isolation module, and when required, it can do partial decommitment. In Example 7-2, the condition $en[1] \ \& \ en[0]$ is taken directly from the corresponding logic in the netlist as it is expressed as a single entity, outside of the operand isolation module.



Tip

To benefit from the full power of operand isolation optimization, break down conditions in `if` statements if it is feasible to do so.

Example 7-1 Nested enable Statements

```
module a (en,in1, in2, in3, clk, out);
input clk;
input [1:0] en;
input [7:0] in1,in2, in3;
output [7:0] out;
reg [7:0] out,y;

always @(posedge clk)
    if (en[1])
        begin
            y = in2 + in3;
            if (en[0])
                begin
                    out = y-in1;
                end
            end
        end
    end
endmodule
```

Example 7-2 Combined enable Statement

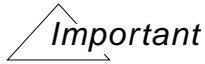
```
module a (en,in1, in2, in3, clk, out);
input clk;
input [1:0] en;
input [7:0] in1,in2, in3;
output [7:0] out;
reg [7:0] out;
wire [7:0] y;

assign y = in2 + in3;
always @(posedge clk)
    begin
        if (en[1] & en[0])
            begin
                out = y-in1;
            end
        end
    end
endmodule
```

Enabling Operand Isolation

To ensure that operand-isolation logic is inserted during synthesis, set the following attribute:

```
set_attribute lp_insert_operand_isolation true /
```



Specify this attribute before the `elaborate` command.

Controlling Naming of Modules, Nets, and Ports

To specify the prefix to be added to all operand-isolation module names, operand-isolation instance names, and ports created during operand-isolation insertion, use the

`lp_operand_isolation_prefix` root attribute. By default, no prefix is added.

Finding Operand-Isolation Logic in the Design Hierarchy

The RC-LP engine creates a separate subdesign for the operand-isolation logic. You can find the subdesigns in the design hierarchy at

```
/designs/design/subdesigns/RC_OI_MODx
```

You can find the instance of each operand-isolation logic subdesign in the design hierarchy at

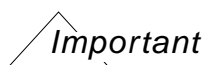
```
/designs/design/instances_hier/*/RC_OI_HIER_INSTx
```

Notes

1. For partially committed operand-isolation logic, the RC-LP engine uses the following naming convention for the instances: `RC_OI_PARTIAL_HIER_INSTx`.
2. You might also see subdesigns such as `RC_OI_CTRL_MOD_x`. These modules contain the control logic which determine when a data signal is used. Instances of these module use the `RC_OI_CTRL_INSTx` naming convention. If the control logic contains only one enable signal that controls the use of the data signal, the `RC_OI_CTRL_MOD` and the `RC_OI_CTRL_INST` are not created.

To find all operand-isolation instances in the hierarchy, you can use the following command:

```
rc:/> find . -inst RC_OI_*HIER_INST*
```



Because the RC-LP engine uses the operand-isolation hierarchy as a vehicle to recognize the operand-isolation logic inserted by RTL Compiler, you should not use the `ungroup` command, as it removes the operand-isolation hierarchy.

Checking System Messages during Optimization

It is important to check the system messages during the synthesis session.

For example, the following messages indicate a normal process. Out of two operand isolation instances, one was committed, and one was partially committed.

```
...
Setting attribute of root '//': 'lp_insert_operand_isolation' = true
Elaborating top-level block 'a' from file 'test.v'.
Performing operand isolation exploration...
    Inserted 3 operand isolation instances in module 'test'
Done elaborating 'test'.
...
    Performing operand isolation commitment...
Operand Isolation Status
=====
Total Operand Isolation instances inserted      3
Total committed                                2
    Fully committed                             2
    Partially committed                         0
Total de-committed                             1
    Due to timing violations                    0
    Due to power violations                     1
....
Mapping test to gates.
    Mapping 'test'...
```

The system messages give an indication of the number of operand isolation instances that

- Were originally inserted during elaboration
- Are committed (fully or partially)
- Are decommitted
- To get more info on the operand isolation instances that are committed or decommitted, set the following root attribute before synthesizing the design:

```
set_attribute information_level 4 /
```

For the previous example, you see the following additional information:

```
    Performing operand isolation commitment...
Info      : Decommitting (deleting) operand isolation instance. [POPT-208]
           : RC_OI_HIER_INST10 does not save power
           :
Info      : Committing Operand Isolation instance. [POPT-207]
           : RC_OI_HIER_INST does not violate timing and it saves power.
           :
Info      : Committing Operand Isolation instance. [POPT-207]
           : RC_OI_HIER_INST9 does not violate timing and it saves power.
Operand Isolation Status
=====
....
```

Reporting Operand Isolation Information

To report operand isolation instances in the design, use the [report operand isolation](#) command. This command provides capabilities for

- [Reporting Summary Information](#)
- [Reporting Detailed Information on Operand-Isolation Instances](#)

The example reports shown in this section use Example [7-3](#).

Example 7-3 Inserting Operand Isolation

RTL Code

```
module test (en,in1, in2, in3, clk, out);
input clk;
input [1:0] en;
input [7:0] in1,in2, in3;
output [7:0] out;
reg [7:0] out,y;

always @(posedge clk)
    if (en[1])
        begin
            y = in2 + in3;
            if (en[0])
                begin
                    out = y-in1;
                end
            end
        end
endmodule
```

Script

```
set_attr library slow.lib

set_attr lp_insert_operand_isolation true /
read_hdl test.v
elaborate

define_clock -name CLK -p 2000 [clock_ports]
external_delay -input 0 -clock CLK [find / -port ports_in/*]
external_delay -input 500 -clock CLK [find / -port ports_in/en[0]]
external_delay -output 6000 -clock CLK [find / -port ports_out/*]

set_attr information_level 4

synthesize -to_map
```

Low Power in Encounter RTL Compiler

Operand Isolation

Reporting Summary Information

- To print a summary report of the operand-isolation logic inserted in the design, enter:

```
rc:/> report operand_isolation
```

Note: You need to be at the design level in the design hierarchy to use this command.

The following example shows the operand-isolation report for [Example 7-3](#) on page 182, after elaboration.

```
=====
Generated by:          RTL Compiler-D (RC) version
Generated on:          date
Module:                a
Technology library:    slow 1.5
Operating conditions:  slow (balanced_tree)
Wireload mode:        segmented
=====

Operand Isolation Instances
-----
Module  Operand Isolation Instance  Width  Isolated Instance  Control Inputs
-----
test    RC_OI_HIER_INST                8      add_11_15          en[1], en[0]
        RC_OI_HIER_INST10            8      sub_14_18          en[1], en[0]
        RC_OI_HIER_INST9             8      add_11_15          en[1], en[0]
-----
Total   3
=====
```

where

- *Width* is the width of the data bus input of the operand-isolation instance
- *Isolated instance* is the datapath instance whose input is isolated.
- *Control inputs* lists the control signals that are inputs to the operand-isolation instance

Low Power in Encounter RTL Compiler

Operand Isolation

Reporting Detailed Information on Operand-Isolation Instances

- To get more detailed information on an inserted operand-isolation instance, enter:

```
rc:/> report operand_isolation -oi_instance name
```

The following example shows a detailed report for the specified operand-isolation instance for [Example 7-3](#) on page 182, after synthesis:

```
rc:/designs/test/> report operand -oi_instance RC_OI_HIER_INST
```

```
=====
...
=====
```

Operand Isolation Instance : RC_OI_HIER_INST

```
-----
Module:                test (test)
Isolated Instance:     (flattened)
Control Inputs:
  RC_OI_CTRL_PORT      =      en[1] ({/designs/a/ports_in/en[1]})
                        TCF = (0.50000, 0.020000/ns)
Data Inputs:
  RC_OI_DATA_PORT      =      in2[7] ({/designs/a/ports_in/in2[7]})
                        in2[6] ({/designs/a/ports_in/in2[6]})
                        in2[5] ({/designs/a/ports_in/in2[5]})
                        in2[4] ({/designs/a/ports_in/in2[4]})
                        in2[3] ({/designs/a/ports_in/in2[3]})
                        in2[2] ({/designs/a/ports_in/in2[2]})
                        in2[1] ({/designs/a/ports_in/in2[1]})
                        in2[0] ({/designs/a/ports_in/in2[0]})
Outputs:
  RC_OI_OUT_PORT       =      n_71 (/designs/test/i..omb/g40/pins_in/B0)
                        n_72 (/designs/test/i..comb/g49/pins_in/A)
                        n_73 (/designs/test/i..comb/g43/pins_in/A)
                        n_74 (/designs/test/i..comb/g50/pins_in/A)
                        n_75 (/designs/test/i..comb/g46/pins_in/A)
                        n_76 (/designs/test/i..comb/g44/pins_in/A)
                        n_77 (/designs/test/i..mb/g39/pins_in/A1N)
                        n_78 (/designs/test/i..comb/g30/pins_in/A)
```

The report shows

- The isolated instance was *flattened*
- The control inputs, data inputs and outputs of the operand-isolation instance
- The switching activities for the control input

Limitations

Currently, operand-isolation logic cannot be inserted in a generic or mapped netlist.

Low Power in Encounter RTL Compiler

Operand Isolation

Leakage Power Optimization

- [Introduction](#) on page 188
- [Prerequisites](#) on page 191
- Tasks
 - [Enabling Leakage Power Optimization](#) on page 192
 - [Selecting Leakage Power Optimization Effort when Using Multiple Vth Libraries](#) on page 193
 - [Controlling Power Optimization](#) on page 194
 - [Preventing Leakage Power Optimization](#) on page 195
 - [Checking System Messages during Optimization](#) on page 196
 - [Reporting Leakage Power](#) on page 198

Introduction

Leakage power is the power dissipated by current leaks in the transistors. The leakage power is usually modelled in the library as a constant. However, some libraries specify cell leakage power as a function of the input state to model leakage power more accurately. In that case, the leakage power is a function of the pin switching activities. For more information on how the tool determines the leakage power, refer to [Leakage Power Calculation](#) in [Chapter 3, “Power Analysis Concepts.”](#)

[Prerequisites](#) provides more details on the required library information.

By giving control over the effort used to propagate the switching activities, the RC-LP engine provides the opportunity to make tradeoffs between accuracy and performance (run time or memory usage). For more information, refer to [Propagating Net Switching Activities](#) in [Chapter 4, “Providing Switching Activity Information.”](#)

The availability of two or more threshold voltages on the same chip (multiple-vt process) provides the opportunity to make trade-offs between power and performance. If you load multiple libraries containing cells with different threshold voltages, the RC-LP engine can make tradeoffs between timing and leakage power during synthesis.

[Figure 8-1](#) on page 189 highlights the tasks you need to add to the generic synthesis flow to use the common leakage power optimization flow starting from RTL.

- [Enabling Leakage Power Optimization](#)
- [Selecting Leakage Power Optimization Effort when Using Multiple Vth Libraries](#) for power-critical designs
- Annotating the switching activities
 - ❑ For a more accurate power optimization, annotate switching activities before synthesis.
 - ❑ For a more accurate power calculation, annotate switching activities before power analysis.
- Specifying the effort to use to propagate the switching activities
- [Reporting Leakage Power](#) after synthesis

Note: If enabled, leakage power optimization occurs automatically during mapping and incremental optimization (`synthesize -to_mapped`).

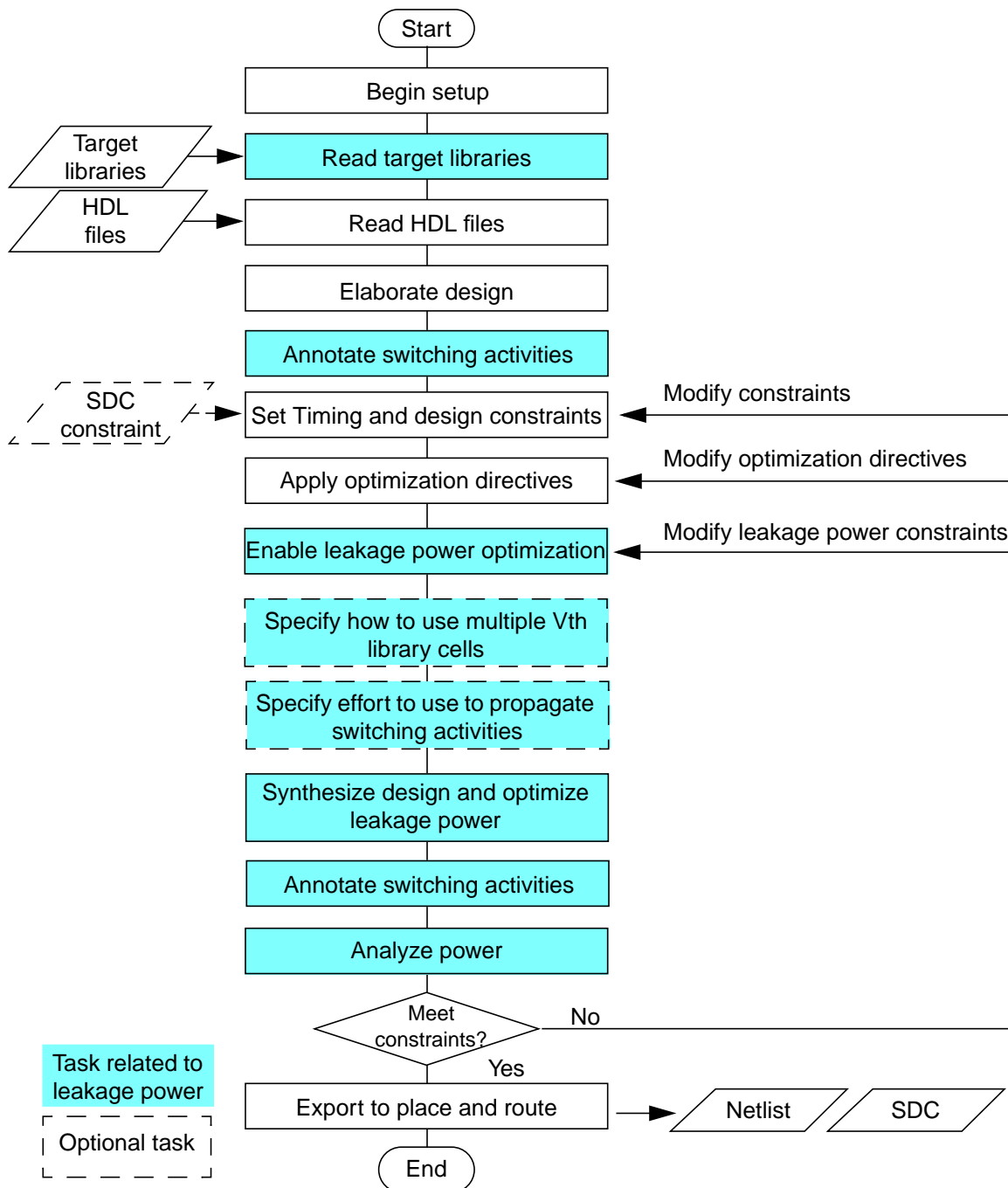
To optimize leakage power in a mapped netlist, refer to [Power Optimization when Starting with Mapped Netlist](#).

Low Power in Encounter RTL Compiler

Leakage Power Optimization

Example 8-1 on page 190 shows an example script for the common leakage power optimization flow.

Figure 8-1 Common Leakage Power Optimization Flow



Low Power in Encounter RTL Compiler

Leakage Power Optimization

Example 8-1 Script for Common Leakage Optimization Flow

```
set_attribute lib_search_path ...
set_attribute hdl_search_path ..

# specify the target libraries, including the multiple vth libraries if any
set_attribute library library_list /

read_hdl design.v
elaborate

# specify timing and design constraints

# (optional) if state-dependent leakage power models:
# run simulation on RTL
# annotate switching activities
read_tcf tcf_file_before_synthesis

# specify leakage power constraint
set_attribute max_leakage_power constraint /designs/design

# (optional) specify leakage power optimization directive
set_attribute lp_multi_vt_optimization_effort medium /designs/design

# (optional) specify effort to use for propagation of switching activities
set_attribute lp_power_analysis_effort medium /

synthesize -to_mapped

# write out netlist for simulation
write -mapped netlist.v

# run simulation on netlist

# annotate switching activities
read_tcf tcf_file_after_synthesis

report power
```

Important

To get the most accurate power optimization and power analysis, you should simulate the design to generate a valid TCF. However, if you want to skip simulation or skip annotating activities, the RC-LP engine will use the default settings and propagate the switching activities from the primary inputs to the primary outputs on those nets that have no switching information asserted.

Prerequisites

During leakage power optimization, the RC-LP engine gets the leakage power cost from the power analysis engine.

If the library uses a *constant* leakage power model, the leakage power optimization is entirely determined by the cell leakage power specified in the technology library. In this case, the cell leakage power can be determined by the following `.lib` attributes:

- `cell_leakage_power`
- `default_cell_leakage_power`
- `default_leakage_power_density`
- `leakage_power_unit`
- `k_process_cell_leakage_power`
- `k_temp_cell_leakage_power`
- `k_volt_cell_leakage_power`

If the library uses a *state-dependent* leakage power model (modeled using the `leakage_power` group in the `cell` group), you should also read in switching activities before optimization to increase the power estimation accuracy.

- For more information on how the leakage power can be defined in the library, refer to [Leakage Power Specification](#) in the *Library Guide for Encounter RTL Compiler*.
- For more information on providing input switching activities, refer to [Chapter 4, “Providing Switching Activity Information.”](#)

As described in [Sources of Switching Activity Information](#), there are several ways to provide switching activity information. However, to obtain accurate power analysis and power optimization results, simulate the design to generate a valid TCF or SAIF file. The RC-LP engine provides a shared library that must be linked with the simulator to generate the TCF files.

Troubleshooting

None of the Libraries Contain Leakage Power Units

If none of the specified technology libraries contain leakage power units, there might be an issue when correlating the results from the different power analysis tools.

If no leakage power units are specified, the RC-LP engine defaults the units to nW.

Enabling Leakage Power Optimization

By default, RTL Compiler optimizes the design for timing and area.

- To enable leakage power optimization, set the following design attribute:

```
set_attribute max_leakage_power power_constraint /designs/design
```

The constraint must be specified in the power units set by the lp_power_unit root attribute.

The max_leakage_power attribute specifies the maximum leakage-power constraint of the design. When the leakage power constraint is set, RTL Compiler performs timing and leakage power optimization simultaneously during mapping and incremental optimization.



Tip

Relax your timing constraints if you want RTL Compiler to achieve the minimum leakage power consumption, regardless of the timing constraints.

Selecting Leakage Power Optimization Effort when Using Multiple V_{th} Libraries

When multiple threshold voltage (V_{th}) libraries are provided, the RC-LP engine can further optimize leakage power by utilizing high V_{th} cells (low leakage power, slow timing performance) along the non-critical timing paths, and low V_{th} cells (higher leakage power, faster timing performance) on timing-critical paths.

Note: The RC-LP engine can optimize the leakage power even if the technology libraries have a single threshold voltage. The only requirement is that you set the leakage power constraint (see [Enabling Leakage Power Optimization](#)). Power savings might be smaller than when providing multiple V_{th} libraries.

Depending on the design requirements, you can choose different optimization effort levels for leakage power optimization with multiple V_{th} libraries.

By default, the RC-LP engine performs a low-effort leakage power optimization. In this case, it uses all V_{th} libraries during each optimization step. The RC-LP engine automatically chooses low V_{th} cells along timing-critical paths and high V_{th} cells along the non-critical timing paths.

If you want to focus on leakage power reduction, and area and run time are not critical, you can perform a medium or high effort level leakage power optimization. When you use a medium effort leakage power optimization, RTL Compiler focuses more on leakage power optimization by using more high V_{th} cells during optimization. When using a high effort level optimization, RTL Compiler restricts the use of low V_{th} cells even more than with a medium effort level optimization.

- To enable a high effort leakage power optimization, set the `lp_multi_vt_optimization_effort` root attribute:

```
set_attribute lp_multi_vt_optimization_effort high /
```

Important

In general, the high effort leakage power optimization using multiple V_{th} libraries results in bigger leakage power savings, but possibly at the cost of an area increase and performance degradation, depending on the design, constraints, and libraries used. Therefore, you should only use the medium and high effort leakage power optimization for power-critical designs if you can afford to trade off area and performance for greater power savings.

Controlling Power Optimization

If you set both leakage power and dynamic power constraints (see [Enabling Dynamic Power Optimization](#)), by default the RC-LP engine gives a higher priority to the leakage power constraint than to the dynamic power constraint. If the total leakage power is larger than the value specified by the `max_leakage_power` constraint, the RC-LP engine optimizes the leakage power first. Otherwise, the RC-LP engine optimizes the dynamic power.

You can control power optimization in two ways:

- [Reducing Dynamic Power First](#)
- [Using Weight Factors](#)

Reducing Dynamic Power First

- To force the RC-LP engine to reduce the dynamic power before optimizing the leakage power, set the following design attribute:

```
set_attribute lp_optimize_dynamic_power_first true /designs/design
```

Using Weight Factors

- To let the RC-LP engine optimize the leakage and dynamic power simultaneously, set the following design attribute:

```
set_attribute lp_power_optimization_weight weight /designs/design
```

If the weight factor is set to w , where w is a floating number between 0 and 1, the RC-LP engine uses the following formula to calculate the weighted power:

$$\text{weighted_power} = w \times \text{leakage_power} + (1 - w) \times \text{dynamic_power}$$

where

- $w \times 100$ percent is the weight for the leakage power optimization
- $(1 - w) \times 100$ percent is the weight for the dynamic power.

If you set the weight to 0.5, you specify an equal weight for the leakage and dynamic power. This results in optimizing the total power of the design.

In general, leakage power is mostly consumed when the design is in idle mode and dynamic power is mostly consumed in active mode. If you know your design spends x percentage time in idle mode, you can set *weight* to $x/100$.

Low Power in Encounter RTL Compiler

Leakage Power Optimization

In many cases, there is a big difference between the leakage and dynamic power in the design. For best results, set the weight to a value close to 1.

Note: When you set the `lp_power_optimization_weight`, the RC-LP engine ignores the value of the `lp_optimize_dynamic_power_first` attribute.

Preventing Leakage Power Optimization

If you do not want to use leakage power optimization, set the following design attribute:

```
set_attribute max_leakage_power {} /designs/design
```

This attribute setting removes the constraint. This is also the default setting of the `max_leakage_power` attribute.

Checking System Messages during Optimization

RTL Compiler performs timing and leakage power optimization simultaneously during mapping and incremental optimization as shown in an extract of the log file below:

```
Global mapping status
=====
Operation          Total    Worst    Leakage
                   Area     Neg      Power
                   -----
global_map         2710     -75      14952
                   Path: coeff[4] --> regs/big_normal/data_out_reg[3]/D
fine_map           2348     -66      10612
                   Path: val[2] --> regs/big_normal/data_out_reg[0]/SI
area_map           2281     -66      9859
                   Path: coeff[3] --> regs/big_normal/data_out_reg[3]/D
power_map          2326    -158      7997
                   Path: ctrl/STATE_reg[1]/CK --> regs/big_normal/data_out_reg[7]/SE
power_map          2337     -66      8193
                   Path: val[1] --> regs/big_normal/data_out_reg[4]/D
...

Incremental optimization status
=====
Operation          Total    Worst    Total    - - DRC Totals - -
                   Area     Neg      Neg      Max      Max      Leakage
                   -----
init_power         2560      0        0        0        0      10104
p_rem_buf          2551      0        0        0        0      10047
p_rem_inv          2525      0        0        0        0      10002
p_merge_bi         2506      0        0        0        0       9982
glob_power         2477      0        0        0        0       9387
power_down         2453      0        0        0        0       9087
...

Done mapping top
Synthesis succeeded.
```

where

- `power_map` refers to power-driven global refine mapping
- `init_power` refers to initial power optimization

Low Power in Encounter RTL Compiler

Leakage Power Optimization

- `p_rem_buf` refers to power-driven buffer removal
- `p_rem_inv` refers to power-driven inverter removal
- `p_merge_bi` refers to power-driven merging of buffers and inverters
- `glob_power` refers to power-driven global gate resizing
- `power_down` refers to power-driven local gate resizing

Note: The power numbers shown in the log file are always given in nW. The value of the `lp_power_unit` root attribute does not affect the log file.

Reporting Leakage Power

To report the leakage power, use the following command:

```
report power
  [-hier | -flat [-nworst number]] [-depth number]
  [-sort mode] [instance | net]... [> file]
```

The following example reports power for a multi-bit muxed adder which has two hierarchical instances: *ma0* and *ma1*.

```
rc:/> report power
=====
...
Module:                mult_bit_muxed_add
Technology library:    umcl13u210t2_wc 1.0
Operating conditions:  <nominal> (balanced_tree)
Wireload mode:         enclosed
=====
```

Instance	Cells	Leakage Power (nW)	Dynamic Power (nW)	Total Power (nW)
mult_bit_muxed_add	6	71.146	1578.273	1649.419
ma0	3	35.573	636.271	671.844
ma1	3	35.573	638.738	674.311

For each hierarchical *instance* you find:

- The number of cells in that hierarchical instance
- Leakage Power—total leakage power for the hierarchical instance (in pW).
- Dynamic Power—the sum of the total internal cell power for the hierarchical instance and the net power of all nets dissipated in the hierarchical instance (in nW).
- Total Power—the sum of the leakage power and the dynamic power of the hierarchical instance (in nW).

To only report the total leakage power of the design, use the following command:

```
get_attribute lp_leakage_power /designs/design
```

To change the leakage power unit used for reporting, use the following command:

```
set_attribute lp_power_unit power_unit /
```

For more information on power analysis, refer to [Chapter 5, “Power Analysis.”](#)

Dynamic Power Optimization

- [Introduction](#) on page 200
- [Prerequisites](#) on page 203
- Tasks
 - [Enabling Dynamic Power Optimization](#) on page 204
 - [Controlling Power Optimization](#) on page 205
 - [Preventing Dynamic Power Optimization](#) on page 205
 - [Checking System Messages during Optimization](#) on page 206
 - [Reporting Dynamic Power](#) on page 207
- [Troubleshooting](#) on page 209

Introduction

Dynamic power is the power dissipated by an instantaneous short-circuit connection between the voltage supply and the ground when the gate transitions, and the switching power dissipated when charging or discharging internal capacitances. Consequently, the dynamic power is a function of the input slew, output load capacitance, and input switching activities. For more information on how the tool determines the internal (dynamic) power, refer to [Internal Cell Power Calculation](#) in [Chapter 3, “Power Analysis Concepts.”](#)

[Prerequisites](#) provides more details on the required library information.

The RC-LP engine uses the RTL Compiler timing engine to obtain accurate slew and load capacitance information during optimization.

By giving control over the effort used to propagate the switching activities, the RC-LP engine provides the opportunity to make trade-offs between accuracy and performance (run time or memory usage). For more information, refer to [Propagating Net Switching Activities](#) in [Chapter 4, “Providing Switching Activity Information.”](#)

[Figure 9-1](#) on page 201 highlights the tasks you need to add to the generic synthesis flow to use the dynamic power optimization flow.

- [Enabling Dynamic Power Optimization](#)
- Annotating the switching activities
 - ❑ For a more accurate power optimization, annotate switching activities before synthesis.
 - ❑ For a more accurate power estimation, annotate switching activities before power analysis.
- Specifying the effort to use to propagate the switching activities
- [Reporting Dynamic Power](#) after synthesis

Note: If enabled, dynamic power optimization occurs automatically during mapping and incremental optimization (`synthesize -to_mapped`).

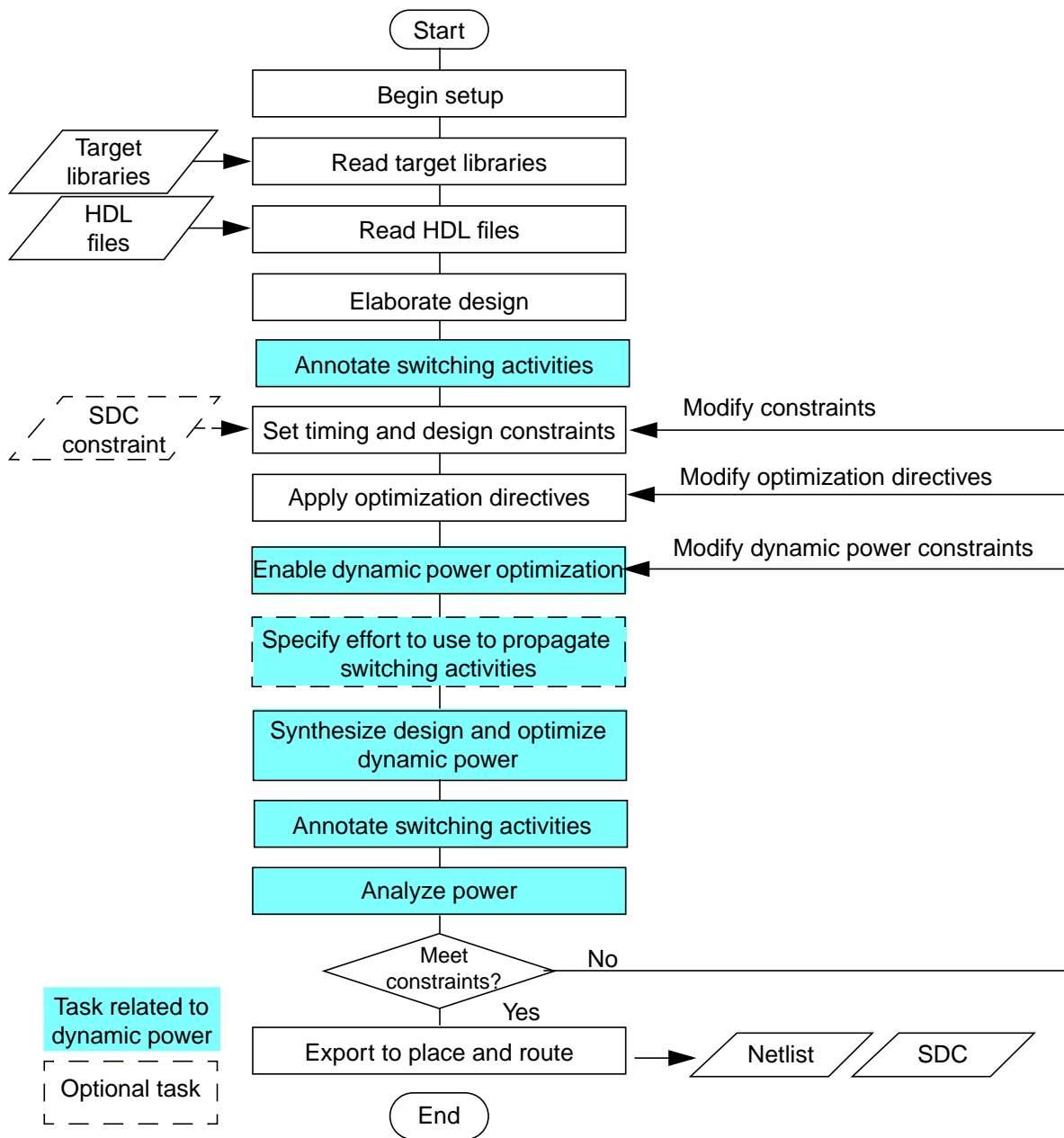
To optimize dynamic power in a mapped netlist, refer to [Power Optimization when Starting with Mapped Netlist](#).

[Example 9-1](#) on page 202 shows an example script for the recommended dynamic flow starting from RTL.

Low Power in Encounter RTL Compiler

Dynamic Power Optimization

Figure 9-1 Recommended Dynamic Power Optimization Flow



Example 9-1 Script for Recommended Dynamic Optimization Flow

```
set_attributer lib_search_path ...
set_attribute hdl_search_path ..

# specify the target libraries
set_attribute library library_list /

read_hdl design.v
elaborate

# run simulation on RTL
# annotate switching activities
read_tcf tcf_file_before_synthesis

# specify timing and design constraints
# specify dynamic power constraint
set_attribute max_dynamic_power constraint /designs/design
# (optional) specify effort for propagation of switching activities
set_attribute lp_power_analysis_effort medium /

synthesize -to_mapped

# write out netlist for simulation
write -mapped netlist.v

# run simulation on netlist
# annotate switching activities
read_tcf tcf_file_after_synthesis

report power
```

Important

To get the most accurate power optimization and power analysis, you should simulate the design to generate a valid TCF. However, if you want to skip simulation or skip annotating activities, the RC-LP engine will use the default settings and propagate the switching activities from the primary inputs to the primary outputs on those nets that have no switching information asserted.

Prerequisites

During dynamic power optimization, the RC-LP engine gets the dynamic power cost from the internal power analysis engine. The internal cell power is a function of the input slew, output load capacitance and the input switching activities.

- For details about internal cell power modeling, refer to [Internal Power Specification](#) in the *Library Guide for Encounter RTL Compiler*.
- The RC-LP engine uses the RTL Compiler timing engine to obtain accurate slew and load capacitance information during optimization.
- For more information on providing input switching activities, refer to [Chapter 4, “Providing Switching Activity Information.”](#)

As described in [Sources of Switching Activity Information](#), there are several ways to provide switching activity information. However, to obtain accurate power analysis and power optimization results, simulate the design to generate a valid TCF or SAIF file. The RC-LP engine provides a shared library that must be linked with the simulator to generate the TCF files.

The switching activities are updated incrementally during optimization by the RC-LP engine.

Enabling Dynamic Power Optimization

By default, RTL Compiler optimizes the design for timing and area.

- To enable dynamic power optimization, set the following design attribute:

```
set_attribute max_dynamic_power power_constraint /designs/design
```

The constraint must be specified in the power units set by the lp_power_unit root attribute.

The max_dynamic_power attribute specifies the maximum dynamic-power constraint of the design. When this constraint is set, the RC-LP engine performs timing and dynamic power optimization simultaneously at all optimization stages: that is, during global mapping, refine mapping and incremental mapping.

- During global and refine mapping, logic on the non-critical paths is mapped to gates with less dynamic power dissipation.
- During incremental mapping, the RC-LP engine performs additional transformations including gate upsizing or down sizing, buffer removal, logic restructuring to further reduce dynamic power dissipation of the design.



Tip

Relax your timing constraints if you want RTL Compiler to achieve the minimum dynamic power consumption, regardless the timing constraints.

Controlling Power Optimization

If you set both leakage power (see [Enabling Leakage Power Optimization](#)) and dynamic power constraints, by default the RC-LP engine gives a higher priority to the leakage power constraint than to the dynamic power constraint.

You can control power optimization in two ways:

- [Reducing Dynamic Power First](#) on page 194
- [Using Weight Factors](#) on page 194

Preventing Dynamic Power Optimization

If you do not want to use dynamic power optimization, set the following design attribute:

```
set_attribute max_dynamic_power {} /designs/design
```

This attribute setting removes the constraint.

Checking System Messages during Optimization

RTL Compiler performs timing and dynamic power optimization simultaneously during mapping and incremental optimization as shown in an extract of the log file below:

Global mapping status

```
=====
```

Operation	Total Area	Worst Neg Slack	Switching Power
global_map	2777	0	651204
fine_map	2667	0	651012
area_map	2563	0	636697
power_map	2537	0	618910
power_map	2508	0	617277

...

Incremental optimization status

```
=====
```

Operation	Total Area	Worst Neg Slack	Total Neg Slack	- - DRC Max Trans	Totals - - Max Cap	Switching Power
...						
p_rem_buf	2503	0	0	0	0	617203
p_rem_inv	2472	0	0	0	0	616922
p_merge_bi	2425	0	0	0	0	616379
glob_power	2362	0	0	0	0	613260
power_down	2331	0	0	0	0	611549

....

Done mapping top
Synthesis succeeded.

where

- **power_map** refers to power-driven global refine mapping
- **init_power** refers to initial power optimization
- **p_rem_buf** refers to power-driven buffer removal
- **p_rem_inv** refers to power-driven inverter removal
- **p_merge_bi** refers to power-driven merging of buffers and inverters
- **glob_power** refers to power-driven global gate resizing
- **power_down** refers to power-driven local gate resizing

Note: The power numbers shown in the log file are always given in nW. The value of the `lp_power_unit` root attribute does not affect the log file.

Reporting Dynamic Power

To report the dynamic power, use the following command:

```
report power
  [-hier | -flat [-nworst number]] [-depth number]
  [-sort mode] [instance | net]... [-verbose] [> file]
```

The following example reports power for a multi bit muxed adder which has two hierarchical instances: `ma0` and `ma1`.

```
rc:/> report power
=====
...
Module:                mult_bit_muxed_add
...
=====
```

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
mult_bit_muxed_add	6	71.146	1578.273	1649.419
ma0	3	35.573	636.271	671.844
ma1	3	35.573	638.738	674.311

For each hierarchical *instance* you find:

- The number of cells in that hierarchical instance
- Leakage Power—total leakage power for the hierarchical instance (in nW).
- Dynamic Power—the sum of the total internal cell power for the hierarchical instance and the net power of all nets dissipated in the hierarchical instance (in nW).

$$\text{Dynamic Power} = \text{Internal Power} + \text{Net Power}$$

- Total Power—the sum of the leakage power and the dynamic power of the hierarchical instance (in nW).

To see the components of the dynamic power for each instance, you can use the `-verbose` option with the `report power` command.

Low Power in Encounter RTL Compiler

Dynamic Power Optimization

The following report shows the components of the dynamic power for the previous example.

```
rc:/designs/mult_bit_muxed_add> report power -verbose
```

```
=====
...
=====
```

Instance	Cells	Leakage Power(nW)	Internal Power(nW)	Net Power(nW)	Dynamic (Int+Net) Power(nW)	Total (Leak+Dyn) Power(nW)
mult_bit_muxed_add	6	71.146	1069.140	509.134	1578.273	1649.419
ma0	3	35.573	533.045	103.226	636.271	671.844
ma1	3	35.573	536.095	102.643	638.738	674.311

To only report the total dynamic power of the design, use the following command:

```
get_attribute lp_internal_power /designs/design
```

To change the power unit used for reporting, use the following command:

```
set_attribute lp_power_unit power_unit /
```

For more information on power analysis, refer to [Chapter 5, "Power Analysis."](#)

Troubleshooting

Clock Gating Yields Negligible Dynamic Power Reduction

If the switching power of your design improves only slightly after inserting clock-gating logic, even though most flops are gated, you can use the following debugging techniques to find the probable cause.

- Make sure you analyze the power before and after clock gating using the same method to annotate the switching activities.
 - If you do not use TCF files, make sure you use the same default settings for the switching activities.
 - If you use a TCF file, you can use the same TCF file if you compare the power results within the same RTL Compiler session. If you compare power results in different sessions, make sure you generate the TCF file for the netlist in either case.
- Check if the switching activities (probability) on the enable pins is close to 1. Followings are the commands:

Find the flop that consumes the most internal power by finding the clock-gating instance (RC_CG_HIER_INST) that consumes the most internal power:

```
report power -nworst 10 -sort internal [find / -inst RC_CG_HIER_INST/*]
```

Check the probability using the following command:

```
report power [get_attr net \  
[find [find / -inst worst_cg_hier_inst ] -pin pins_in/enable]]
```

Note: If the probability of an enable pin is almost 1, you do not save much power by inserting a clock-gating instance.

- Check the total instance count if you performed clock-gating insertion starting from RTL.
- Make sure you use the same power analysis effort: set the `lp_power_analysis` attribute to the same value (low, medium or high) in both cases.

If there is no difference, find the leaf instance that consumes the most internal power. Make sure it is not a pad, RAM, or ROM that prevents you from observing the internal power difference in the normal standard cells by using the following commands:

```
report power -nworst 10 -sort internal [find / -inst instances_seq/*]  
report power -nworst 10 -sort internal [find / -inst instances_comb/*]
```

Low Power in Encounter RTL Compiler

Dynamic Power Optimization

Using CPF for Multiple Supply Voltage Designs

- [Overview](#) on page 212
- [CPF File for MSV Design](#)
- [MSV Information in the Design Information Hierarchy](#) on page 215
- [Flow Steps](#) on page 217
 - ❑ [Read Target Libraries](#) on page 219
 - ❑ [Read CPF File](#) on page 220
 - ❑ [Check the CPF File](#) on page 221
 - ❑ [Set Timing, Design and Power Constraints](#) on page 222
 - ❑ [Apply Clock-Gating and Optimization Directives](#) on page 222
 - ❑ [Annotate Switching Activities](#) on page 223
 - ❑ [Estimate RTL Power](#) on page 223
 - ❑ [Synthesize Design](#) on page 223
 - ❑ [Execute the CPF File](#) on page 224
 - ❑ [Verify Added Power Logic](#) on page 225
 - ❑ [Run Incremental Optimization](#) on page 226
 - ❑ [Annotate switching activities](#) on page 226
 - ❑ [Analyze Power](#) on page 226
 - ❑ [Analyze Design](#) on page 227
 - ❑ [Export to Place and Route](#) on page 230

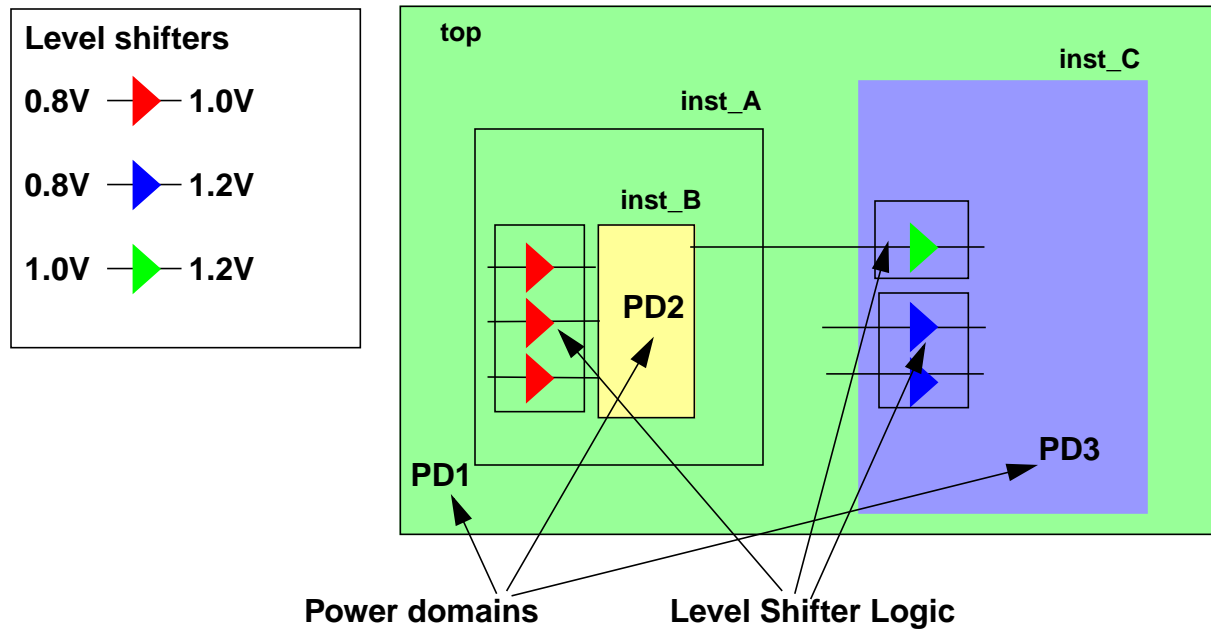
Overview

A Multiple Supply Voltage (MSV) design uses multiple supply voltages for the core logic. In Figure 10-1 the top design and instance `inst_A` operate on voltage `VDD1`, while instance `inst_B` operates on voltage `VDD2` and instance `inst_C` operates on voltage `VDD3`.

A portion of the design that operates at the same operating voltage (that is, uses the same *main* power supply) belongs to the *power domain* that corresponds to that operating voltage.

Figure 10-1 Example of MSV Design

Operating Voltage (OV)	Instances Operating on OV	Libraries characterize for OV	Library Set	Power Domains for OV
VDD1: 0.8	top, inst_A	lib1 lib2	set1	PD1
VDD2: 1.0	inst_B	lib3	set2	PD2
VDD3: 1.2	inst_C	lib4	set3	PD3



Because libraries are characterized for a particular set of operating conditions, blocks operating on a particular voltage need to use libraries that were characterized for that particular voltage. You can group libraries that were characterized for the same nominal operating conditions in a *library set*. In Figure 10-1, a library set was created for each voltage. Libraries `LIB1` and `LIB2` were associated with library set `set1`, and so on.

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

To pass signals between portions of the design that operate on different voltages, *level shifters* are needed. Level shifter cells are usually characterized for the different voltages they operate between, you'll therefore need to identify level shifter cells for all appropriate voltage (power domain) pairs.

The majority of cells in a power domain are driven by the same power supply, except for some special cells, such as level shifters which are driven by multiple power supplies.

The Common Power Format (CPF) file captures the information described above in a text file. The [CPF File for MSV Design](#) shows how this information is described.

[MSV Information in the Design Information Hierarchy](#) shows where the additional MSV-specific information is stored.

[Flow Steps](#) shows where in the flow you need to read and execute the CPF file.

CPF File for MSV Design

The CPF file for the example in [Figure 10-1](#) on page 212 is shown in [Figure 10-2](#). [Creating a CPF File for an MSV Design](#) in the *Common Power Format User Guide* shows how to create a CPF file for an MSV design.

The comments in the example below show the type of information that is specified.

Figure 10-2 CPF File for MSV Example

```
# group libraries characterized for same operating condition
define_library_set -name set1 -libraries {lib1 lib2}
define_library_set -name set2 -libraries lib3
define_library_set -name set3 -libraries lib4

# identify level shifter cells for the voltage combinations needed
define_level_shifter_cell -cells LVLLHEHX* \
    -input_voltage_range 0.8 -output_voltage_range 1.0 \
    -output_power_pin VDD -ground GND -direction up
define_level_shifter_cell -cells LVLLHX* \
    -input_voltage_range 0.8 -output_voltage_range 1.2 \
    -output_power_pin VDD -ground GND -direction up
define_level_shifter_cell -cells LVLLHELX* \
    -input_voltage_range 1.0 -output_voltage_range 1.2 \
    -output_power_pin VDD -ground GND -direction up

# identify the design for which the CPF file is created
set_design top
```

Low Power in Encounter RTL Compiler

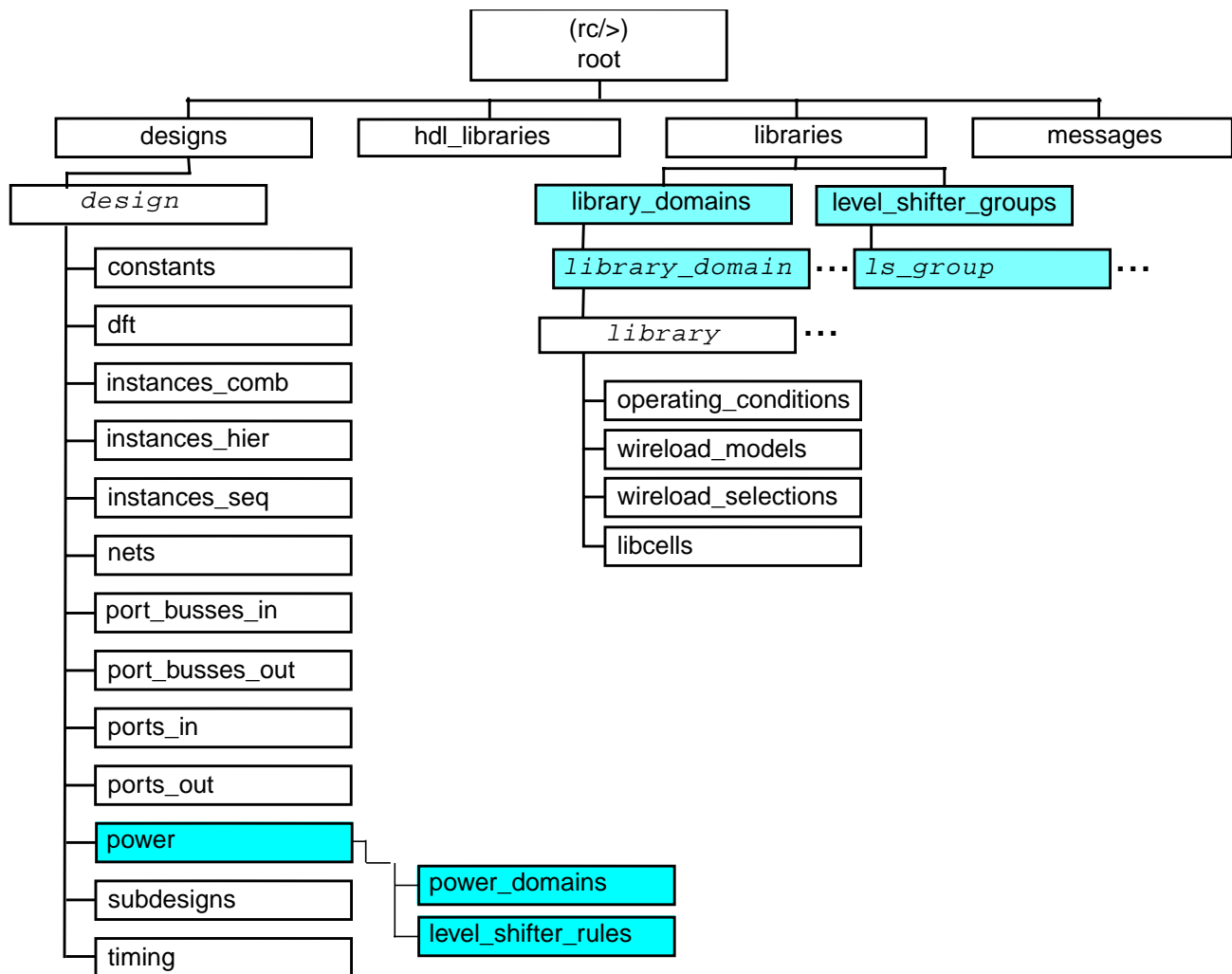
Using CPF for Multiple Supply Voltage Designs

```
# identify portions of the design that operate on the same voltage
create_power_domain -name PD1 -default
create_power_domain -name PD2 -instances instance_B
create_power_domain -name PD3 -instances instance_C
# specify the operating voltages used in the design
create_nominal_condition -name low -voltage 0.8
create_nominal_condition -name medium -voltage 1.0
create_nominal_condition -name high -voltage 1.2
# associate the nominal conditions with the power domains
create_power_mode -name PM -domain_conditions {PD1@low PD2@medium PD3@high} \
-default
# specify which library set to use for a specific nominal condition
update_nominal_condition -name low -library_set set1
update_nominal_condition -name medium -library_set set2
update_nominal_condition -name high -library_set set3
# create level-shifter rules
create_level_shifter_rule -name lsr1 -from PD1 -to PD2
create_level_shifter_rule -name lsr2 -from PD2 -to PD3
create_level_shifter_rule -name lsr3 -from PD1 -to PD3
# specify the targets for leakage and dynamic power
set_power_target -leakage 30 -dynamic 250
# specify the timing constraints and activity information
update_power_mode -name PM -sdc_files top.sdc
update_power_mode -name PM -activity_file top.tcf -activity_file_weight 100
# specify where level shifters must be created
update_level_shifter_rules -names lsr1 -location from
update_level_shifter_rules -names {lsr2 lsr3} -location to
#
update_power_domain -name PD1 -internal_power_net VDD1
update_power_domain -name PD2 -internal_power_net VDD2
update_power_domain -name PD3 -internal_power_net VDD3
# indicate when the power information for the design ends
end_design
```

MSV Information in the Design Information Hierarchy

RTL Compiler stores the original design data along with additional information in the CPF file in the design information hierarchy in the form of attributes. Figure 10-3 highlights the MSV information in the design information hierarchy.

Figure 10-3 Design Information Hierarchy



This section uses the comments in the CPF file shown in Figure 10-2 on page 213 to show how the information is stored in the design hierarchy.

group libraries characterized for same operating condition

The RC-LP engine creates a `library_domains` directory under the `libraries` directory. For each *library set* defined in the CPF file, it creates a subdirectory with that name in the `library_domains` directory. The libraries in each set are stored in the corresponding library domain.

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

identify level shifter cells for the voltage combinations needed

The RC-LP engine *identifies* the correct level shifter group for each library cell defined by a `define_level_shifter_cell` command. All library cells that have the same input and output voltage range specification and that belong to the same library are grouped in the same level shifter group.

identify the design for which the CPF file is created

The RC-LP engine verifies if the CPF file is defined for the design that is currently loaded.

identify portions of the design that operate on the same voltage

The RC-LP engine creates a `power` directory under the design directory with a subdirectory named after each *power domain* that was defined in the CPF file.

For any instance associated with a specific power domain, the RC-LP engine sets the following instance attribute:

```
set_attribute power_domain power_domain instance
```

specify the operating voltages used in the design

associate the nominal conditions with the power domains

specify which library set to use for a specific nominal condition

The information in these three steps has this effect:

- Associates a library set characterized for a nominal condition with the appropriate power domain.
- Sets the `voltage_operating_condition` attribute to the specified value.

```
set_attribute voltage voltage library_set/library/operating_conditions/  
library
```

create level-shifter rules

The RC-LP engine creates a `level_shifter_rules` directory under the `power` directory and with a subdirectory named after each *level_shifter rule* defined in the CPF file.

specify the targets for leakage and dynamic power

When you specify these targets, the RC-LP engine sets the following design attributes:

```
set_attribute max_leakage_power power_constraint /designs/design
```

```
set_attribute max_dynamic_power power_constraint /designs/design
```

specify the timing constraints and activity information

The information specified in the SDC file is stored in the timing directories.

The information specified in a TCF or SAIF file is annotated on pins, ports, and nets.

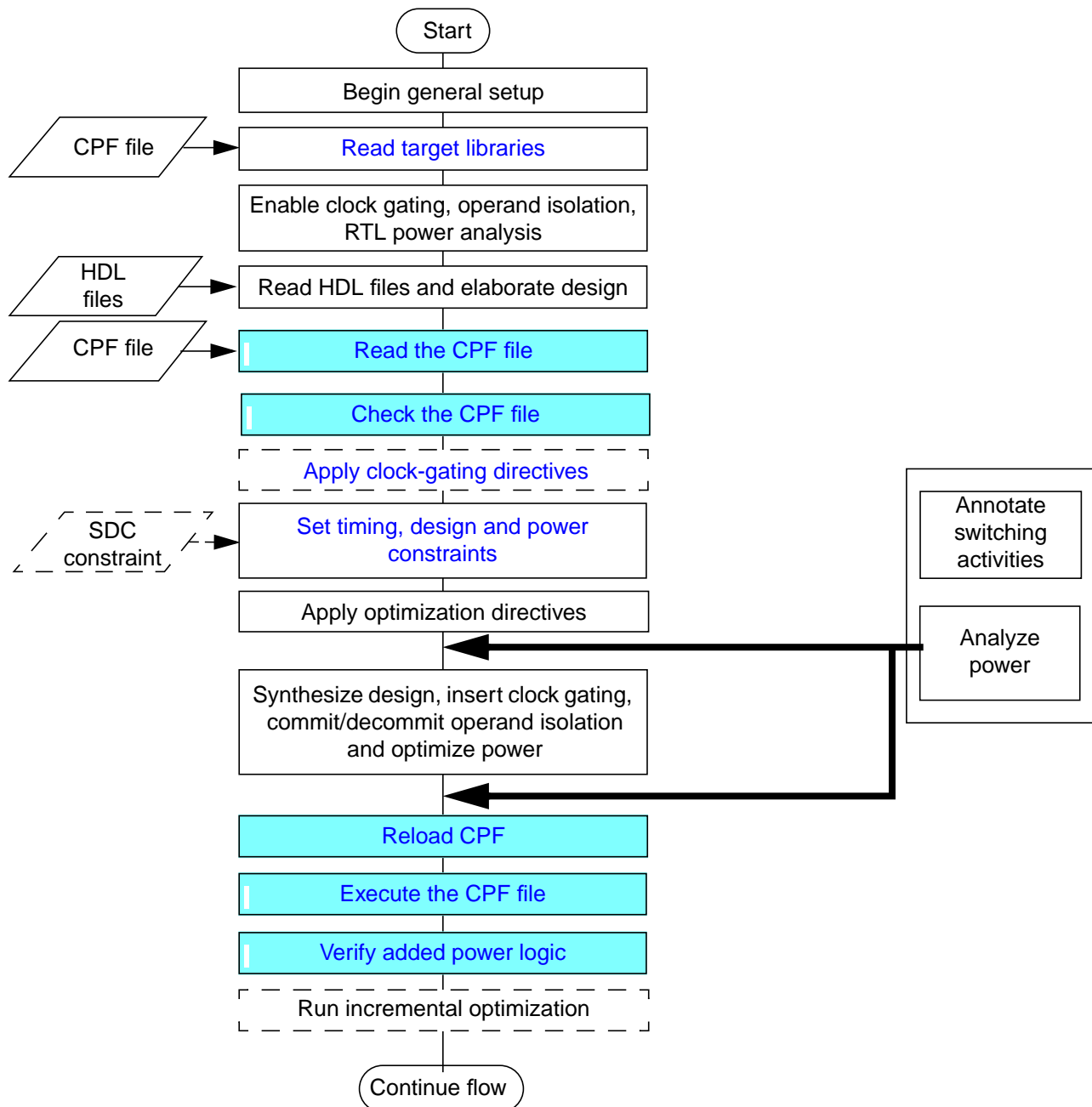
specify where level shifters must be created

This information is used when the CPF file is executed and level shifters are inserted.

Flow Steps

Figure 10-4 shows the recommended flow and highlights the tasks you need to add to the low power synthesis flow to use multiple supply voltages in the design. A script for this flow, further referred to as the *CPF-MSV* flow, is shown in [Example 10-1](#) on page 218.

Figure 10-4 Top-Down MSV Low Power Flow with CPF



Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

Example 10-1 Script for Common Top-Down CPF-MSV Flow

```
# general setup
#-----
set_attribute hdl_search_path ..

# specify the target libraries for elaboration
#-----
read_cpf -library cpf_file
check_library

# enable clock-gating, operand isolation, and RTL power analysis
#-----

set_attribute lp_insert_clock_gating true /
set_attribute lp_insert_operand_isolation true /
set_attr hdl_track_filename_row_col true /

# read and elaborate the design
#-----
read_hdl design.v
elaborate

# read the cpf file
#-----
read_cpf cpf_file
set_attribute lec_executable /tools/lec_version/bin/lec /
check_cpf

# apply clock-gating directives
#-----
set_attribute lp_clock_gating_module cg_module subdesign_list
...
set_attribute lp_clock_gating_cell libcell subdesign_list
...

# read SDC file if file is not read via the CPF file
#-----
read_sdc sdc_file

# read switching activity file if file is not read via the CPF file
#-----
read_tcf tcf_file

#synthesize the design
#-----
synthesize -to_mapped

# execute the CPF file
#-----
commit_cpf

report_level_shifter -hier -detail
verify_power_structure

#run incremental optimization
#-----
synthesize -incremental

# analyze design
#-----
report timing
report gates
report power

# export design
#-----
write_encounter design [design] ...
```

Low Power in Encounter RTL Compiler

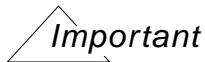
Using CPF for Multiple Supply Voltage Designs

Because the generic flow was described in [Chapter 2, “Recommended Flow,”](#) this section will focus on the additional steps and some of the steps modified for the CPF-MSV flow.

Read Target Libraries

Read the target libraries needed to elaborate the design.

```
read_cpf -library cpf_file
```



The `read_cpf` command does not use the `lib_search_path` attribute setting to load libraries defined in the CPF file with the `define_library_set` command. Because the CPF file is meant to be read by multiple tools, you must provide either full or relative file name paths in the CPF file.



To specify the file paths in the CPF file, you can use environment variables.

To access Tcl variables defined in the RTL Compiler shell inside the CPF file, you need to add the global name space operator (`::`) in front of the variable. For example.

Assume the following Tcl variable is defined in your script

```
set reset_on_design 1
```

To use this variable inside your CPF file, use:

```
if {$::reset_on_design}
```

At this point the RC-LP engine

- Creates library domains in the design information hierarchy.

The names of the library domains correspond to the names of the library sets defined in the CPF file. The libraries in the library sets are stored in the corresponding library domains.

- Creates the level shifter groups according to the CPF specifications

The RC-LP engine creates a level shifter group for each input and output voltage range pair in the `define_level_shifter_cell` commands to point to the library cells that are defined as level shifters in this category.

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

More specifically, level shifters will be in same level shifter group if the following criteria are met:

- ❑ The level shifters belong to same library domain
- ❑ The level shifters have the same input and output voltage range
- ❑ The level shifters have the same direction, valid location and enable voltage
- ❑ The level shifters have the same functionality

For example, an AND gate can not be grouped with a buffer.

Important

If you specified to use some specific library cell(s) in the `define_xxx_cell` commands but the library cells are marked `dont_use` in the `.lib` file, the RC-LP engine will use the `set_attribute preserve false libcell` and `set_attribute avoid false libcell` commands to try to mark the library cell(s) usable. To prevent this, modify the CPF file and remove those cells from the `define_xxx_cell` commands.

Caution

If you read in libraries to be used for elaboration only using the `library root` attribute, the RC-LP engine will remove these libraries when the CPF file is read in. This implies that any attributes set on these libraries will be lost when reading in the CPF file.

Read CPF File

Capture the power intent for your design.

```
read_cpf cpf_file
```

When reading the CPF file, the tool performs the following actions:

- Creates the power domains in the design information hierarchy.
- Stores the association of the library sets with the power domains.

You can verify the information as follows:

```
get_att library_domain power_domain
```

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

- Stores the association of the power domains with the specified instances in the design information hierarchy.

You can verify the information as follows:

```
rc:/designs/top> get_att power_domain [find / -instance instance]
```

Important

If you marked an instance *preserved*, the power domain that the instance is associated with will still be changed. However, the instance will still be marked preserved even though it will probably be pointing to another library cell in the new power domain. In other words, the `power_domain` attribute has a higher priority than the `preserve` attribute.

If RTL Compiler cannot find the corresponding cell in the libraries that correspond to the new power domain, the instance becomes *unresolved*.

- Reads the SDC constraints file(s).
- Reads the TCF file(s).
- Marks the necessary pins and nets *preserved* to ensure technology mapping respects the power domain boundaries.
- Creates the level-shifter rules directories.

Check the CPF File

After reading the CPF file, check the validity of the CPF rules against the RTL of your design. This enables you to capture any violations of the low power intent of the design early in the design cycle.

`check_cpf`

Note: To run this command you need to have access to version 7.1 (or later) of the Encounter[®] Conformal[®] Low Power software.

Caution

If rule check errors are detected, you need to make the necessary changes to your CPF file before proceeding further with synthesis.

Set Timing, Design and Power Constraints

1. If you did not specify an SDC file in the CPF file, you can provide timing and design constraints at this time.

```
read_sdc sdc_file
report timing -lint
```

2. If you did not specify the targets for leakage and dynamic power optimization in the CPF file, set the following attributes:

```
set_attribute max_leakage_power float /designs/design
set_attribute max_dynamic_power float /designs/design
```

The RC-LP engine continues power optimization until the power of the design is smaller than the specified constraint. If leakage and dynamic power constraints are specified, leakage power optimization is optimized first by default.

3. To control the power optimization, use one of the following design attributes:

```
set_attribute lp_optimize_dynamic_power_first true /designs/design
set_attribute lp_power_optimization_weight float /designs/design
```

For more information on leakage power optimization, refer to [Chapter 8, “Leakage Power Optimization.”](#) For more information on dynamic power optimization, refer to [Chapter 9, “Dynamic Power Optimization.”](#)

Apply Clock-Gating and Optimization Directives

1. Select the clock-gating logic to use. You can either
 - ❑ Specify which clock-gating module or integrated cell to use, by associating either of the following subdesign attributes with the appropriate subdesigns:

```
set_attribute lp_clock_gating_module cg_module subdesign_list
set_attribute lp_clock_gating_cell libcell subdesign_list
```

- ❑ Select the clock-gating cell that is available in each library domain, by setting the following design attributes (or using the default):

```
lp_clock_gating_add_obs_port
lp_clock_gating_add_reset
lp_clock_gating_control_point
lp_clock_gating_style
```

2. To control the clock gating, use the following design attributes:

```
lp_clock_gating_max_flops
lp_clock_gating_min_flops
lp_clock_gating_exclude
lp_clock_gating_extract_common_enable
```

For more information, see [Controlling Insertion of Clock-Gating Logic.](#)

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

In addition to applying design constraints, you may need to use further optimization strategies to get the desired performance goals from synthesis.

For more information on optimization strategies and related commands, see [Defining Optimization Settings](#) in *Using Encounter RTL Compiler*.

Annotate Switching Activities

If you did not specify any activity information using the CPF file, you should annotate switching activities before power analysis for a more accurate power calculation.

For more information, see [Chapter 4, “Providing Switching Activity Information.”](#)

Estimate RTL Power

Note: This corresponds to the *Analyze Power* step before *Synthesize Design*.

- To analyze the power at the RTL level, use the following command:

```
report power -rtl -detail
```

For more information, refer to [RTL Power Analysis](#) in [Chapter 5, “Power Analysis.”](#)

Synthesize Design

- After the constraints and optimizations are set for your design, synthesize your design using the following command:

```
synthesize -to_mapped
```

The different portions of the design that are associated with different power domains will be mapped to the target libraries of those power domains and optimized.

Clock-gating insertion, commitment and decommitment of the operand isolation instances, and leakage and dynamic power optimization occur automatically during mapping and optimization.

Note: Clock-gating insertion is power domain-aware. This implies that a clock-gating instance cannot gate flip-flops across power domains. A clock-gating instance gating the flip-flops in a power domain also belongs to that power domain.



Tip

In addition, if the setting of the `lp_clock_gating_min_flops` attribute can not be satisfied, no clock gating will be inserted.

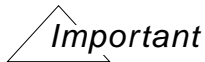
Reload CPF File

The netlist can change when the design is synthesized. New design objects (ports, pins, nets, and instances) can be added (for example, for DFT or for low power). To ensure the proper pin (port) selection for level shifter insertion, you need to reapply the CPF rules.

Execute the CPF File

At this time the RC-LP engine inserts the level shifters.

- To execute the CPF file, use the `commit_cpf` command.



RTL Compiler can only insert level shifters between two power domains if you defined the appropriate level-shifter rules in the CPF file.

RTL Compiler will not insert level shifters on

- ☐ constant nets
 - ☐ dangling pins or nets
 - ☐ multiple-driver nets if all or some drivers reside in different library domains and you request to insert the level shifters in the destination power domain (to domain)
- To report all the level shifters inserted in the design, use the `report_level_shifter` command:

```
report_level_shifter -hier -detail
```

RTL Compiler groups level shifter instances inserted to pass signals between the same domains in a hierarchical instance.

Finding Level Shifters in the Hierarchy

RTL Compiler creates a separate subdesign for the level shifters connecting two library domains. You can find level-shifter subdesigns in the design hierarchy at

```
/designs/design/subdesigns/prefix_MOD*
```

You can find the path to a level-shifter instance in the design hierarchy using the `find` command:

```
find / -instance prefix_HIER_INST*
```

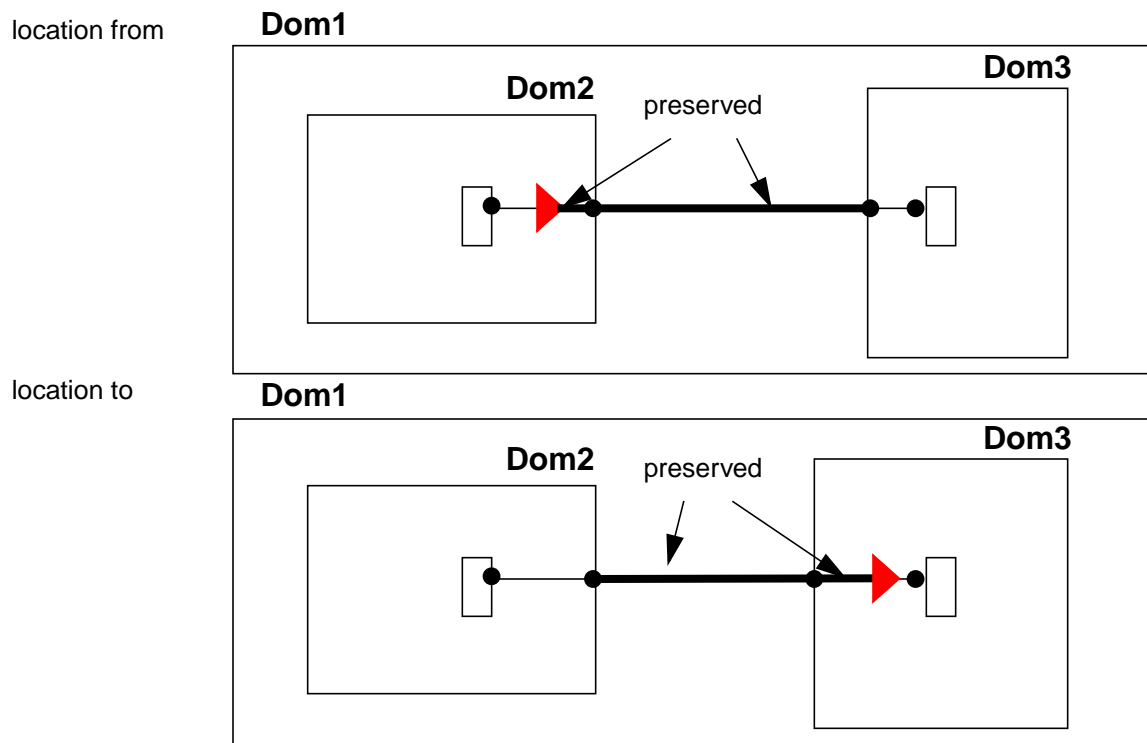
Prefix refers to the prefix you specified using the `-prefix` option of the `update_level_shifter_rules` command. The default prefix is `CPF_LS`.

Preserve Actions

RTL Compiler automatically sets the `preserve` attribute on the level-shifter subdesigns to `size_delete_ok`. However, if you use a dedicated library cell for level shifter insertion, RTL Compiler sets the `preserve` attribute value to `delete_ok` on the level-shifter subdesigns.

When a net crosses multiple library domains, RTL Compiler automatically sets the `preserve` attribute on some portions of the net to prevent buffer insertion as illustrated in Figure 10-5.

Figure 10-5 Preserving Nets to Prevent Buffer Insertion



Verify Added Power Logic

Verifies whether the level shifters that were inserted in the design conform to the level shifter rules in the loaded CPF file. The tool will flag if there are any missing level shifters or if the level shifters are not connected appropriately.

`verify_power_structure -lvl`

Note: To run this command you need to have access to version 7.1 (or later) of the Encounter[®] Conformal[®] Low Power software.

Run Incremental Optimization

Inserting the level shifters can have an impact on the timing.

- To fix any timing issues, run incremental optimization:

```
synthesize -incremental
```

Annotate switching activities

- To get accurate power analysis results, read in the switching activity information generated from the mapped netlist:

```
read_tcf tcf_file_for_mapped_netlist
```

Analyze Power

- Analyze the power using the following command:

```
report power
```

For more information, refer to [Chapter 5, “Power Analysis.”](#)

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

Analyze Design

After synthesizing the design, you can generate detailed timing and area reports using the various `report` commands. For more information, see [Analyze Design](#) in the [Recommended Flow](#).

Most reports reflect information for the library domains (sets in CPF). The following example shows the top of the `report gates` report.

```
rc:/> report gates

=====
Generated by:          RTL Compiler-D (RC) version
Generated on:          date
Module:               top
Library domain:       0.8v
Domain index:         0
Technology library:    lib1_125c 1.1
Operating conditions: lib1_125c (balanced_tree)
Library domain:       levellib
Domain index:         1
Technology library:    ls_0v70_0v80 1.0
Operating conditions: ls_0v70_0v80 (balanced_tree)
Library domain:       0.7v
Domain index:         2
Technology library:    lib2_125c 1.1
Operating conditions: lib2_125c (balanced_tree)
Wireload mode:        enclosed
=====
```

Gate	Instances	Area	Library	Library Domain
AOI21XL	2	11.767	lib1_125c	0.8v
....				
INVL	2	7.060	lib2_125c	0.7v
INVL	3	10.590	lib1_125c	0.8v
LVLLEH8	12	931.946	ls_0v70_0v80	levellib
...				
SDFFRQX1	10	329.476	lib2_125c	0.7v

total	84	1640.320		

Library	Instances	Instances %
ls_0v70_0v80	12	14.3
lib2_125c	12	14.3
lib1_125c	60	71.4

Type	Instances	Area	Area %
sequential	10	329.476	20.1
inverter	5	17.651	1.1
logic	69	1293.193	78.8

total	84	1640.320	100.0

This report shows that RTL Compiler used the same cell `INVL` from two library domains.

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

You can also restrict the report to only give you information for one or more library domains (library sets in CPF).

```
rc:/> report gates -library_domain 0.7v
```

```
=====
...
=====
```

Gate	Instances	Area	Library	Library Domain
INVL	2	7.060	ss_0v70_125c	0.7v
SDFFRQX1	10	329.476	ss_0v70_125c	0.7v
total	12	336.536		

Library	Instances	Instances %
lib2_125c	12	14.3

Type	Instances	Area	Area %
sequential	10	329.476	20.1
inverter	2	7.060	0.4
total	12	336.536	20.5

In the timing report (shown in [Figure 10-6](#) on page 229), the domain index allows you to see how the critical path goes from one library domain to the next.

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

Figure 10-6 Timing Report

```
rc:/> report timing
....
Warning : Possible timing problems have been detected in this design. [TIM-11]
        : The design is 'top'
        : Use 'report timing -lint' for more information.
=====
Generated by:          RTL Compiler-D (RC) version
Generated on:          date
Module:               top
Library domain:       0.8v
Domain index:         0
...
Wireload mode:        enclosed
=====
```

Pin	Type (Domain)	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)	
reg_bank_1							
out_reg[0]/CK				0		0	R
out_reg[0]/Q	SDFFRQX1(0.7v)	2	13.2	330	+1069	1069	F
reg_bank_1/out[0]							
alu_inst/a[0]							
leveler_HIER_INST_13/a[0]							
g40/A					+0	1069	
g40/Y	(P) LVLLHEHX8(levellib)	4	5.1	63	+578	1648	F
leveler_HIER_INST_13/g418_A							
g414/B					+0	1648	
g414/Y	NAND2XL(0.8v)	5	6.8	374	+245	1892	R
g394/A					+0	1892	
g394/Y	INVL(0.8v)	1	1.4	123	+114	2006	F
g391/A					+0	2006	
...							
g359/Y	OAI211XL(0.8v)	1	0.0	204	+211	4005	F
alu_inst/out[4]							
out[4]	out port				+0	4005	F

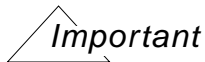
```
-----
Timing slack : UNCONSTRAINED
Start-point  : reg_bank_1/out_reg[0]/CK
End-point    : out[4]
(P) : Instance is preserved
```

Export to Place and Route

- To export the necessary files for the Encounter place and route tool, use the following command:

```
write_encounter design [design] ....
```

This command writes the netlist file, the SDC constraint file, templates for the Encounter config file, Encounter setup file, and Encounter level-shifter table file. The directory and base name of the files are controlled through the `-basename` option of the `write_encounter` command. The default base name is `rc_enc_des`.



This command creates *template* files. It is your responsibility to fill in the remaining information needed to start the Encounter place and route tool.

Encounter Tcl Script Template

The `write_encounter` command creates a setup template, `rc.enc_setup`. This file sets the following information (for MSV):

1. Loads in the Encounter configuration template that it created (see 1 in Figure 10-7).
2. Loads in the original CPF file (see 2 in Figure 10-7) and commits the CPF file.
3. Loads in an MSV-specific script —this file is empty for the CPF-flow.

Figure 10-7 Extract of a Sample Encounter Setup Template File

```
#####
#
# First Encounter setup file
# Created by RTL Compiler (RC) on date time
#
#####

...
# Design Import
#####
...
loadConfig rc_enc_des/rc.conf ← 1

# Mode Setup
#####
source rc_enc_des/rc.mode

# MSV Setup
#####
loadCPF /home/ria/rc_ex/cpf/ls_insert/my.cpf ← 2
commitCPF
source rc_enc_des/rc.msv.tcl
....
```

Encounter Config Template

The `write_encounter` command creates a template for the Encounter config file, `rc.conf`.

The following information is filled in the config file (as shown in Figure 10-8):

- The name of the netlist file (1)
- The type of the netlist file (2)
- The names of the timing libraries (3)
- The name of the constraint file (4)
- The list of the LEF files specified through the `-lef` option or read in during the RTL Compiler session (5)
- The power net names that RTL Compiler created based on the domain names (6). The power and net names are also used in the command file.

Figure 10-8 Sample Encounter Config Template File

```
#####
#
# First Encounter input configuration file
# Created by RTL Compiler (RC) on date time
#
#####
global rda_Input
set cwd lpwd
set rda_Input(ui_netlist) {rc_enc_des/rc.v} ← 1
set rda_Input(ui_netlisttype) {Verilog} ← 2
set rda_Input(ui_settop) {1}
set rda_Input(ui_topcell) {top}
set rda_Input(ui_timelib) {/my_path/lib1_1v08.lib \
...} ← 3
set rda_Input(ui_timingcon_file) {rc_enc_des/rc.sdc} ← 4
set rda_Input(ui_buf_footprint) {BUFX2MTH}
set rda_Input(ui_inv_footprint) {INVX12MTH}
set rda_Input(ui_leffile) {} ← 5
set rda_Input(ui_cts_cell_list) {CLKBUFX8MTH ...}
set rda_Input(ui_core_cntl) {aspect}
set rda_Input(ui_aspect_ratio) {1.0000}
set rda_Input(ui_captbl_file) {}
set rda_Input(ui_defcap_scale) {1.0}
set rda_Input(ui_res_scale) {1.0}
set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_pwrnet) {VDD_PD1 VDD_PD2 ....} ← 6
set rda_Input(ui_gen_footprint) {1}

##### RTL #####
#set rda_Input(ui_netlisttype) {RTL}
```

Low Power in Encounter RTL Compiler

Using CPF for Multiple Supply Voltage Designs

Using CPF for Designs Using Power Shutoff Methodology

- [Overview](#) on page 234
- [CPF File for PSO Design](#)
- [PSO Information in the Design Information Hierarchy](#) on page 239
- [Flow Steps](#) on page 242
 - ❑ [Read Target Libraries](#) on page 244
 - ❑ [Read CPF File](#) on page 245
 - ❑ [Check the CPF File](#) on page 246
 - ❑ [Set Timing, Design and Power Constraints](#) on page 246
 - ❑ [Apply Clock-Gating and Optimization Directives](#) on page 247
 - ❑ [Annotate Switching Activities](#) on page 247
 - ❑ [Estimate RTL Power](#) on page 248
 - ❑ [Synthesize Design](#) on page 249
 - ❑ [Reload CPF File](#) on page 250
 - ❑ [Execute CPF File](#) on page 250
 - ❑ [Verify Added Power Logic](#) on page 254
 - ❑ [Run Incremental Optimization](#) on page 254
 - ❑ [Annotate switching activities](#) on page 254
 - ❑ [Analyze Power](#) on page 254
 - ❑ [Analyze Design](#) on page 256
 - ❑ [Export to Place and Route](#) on page 256

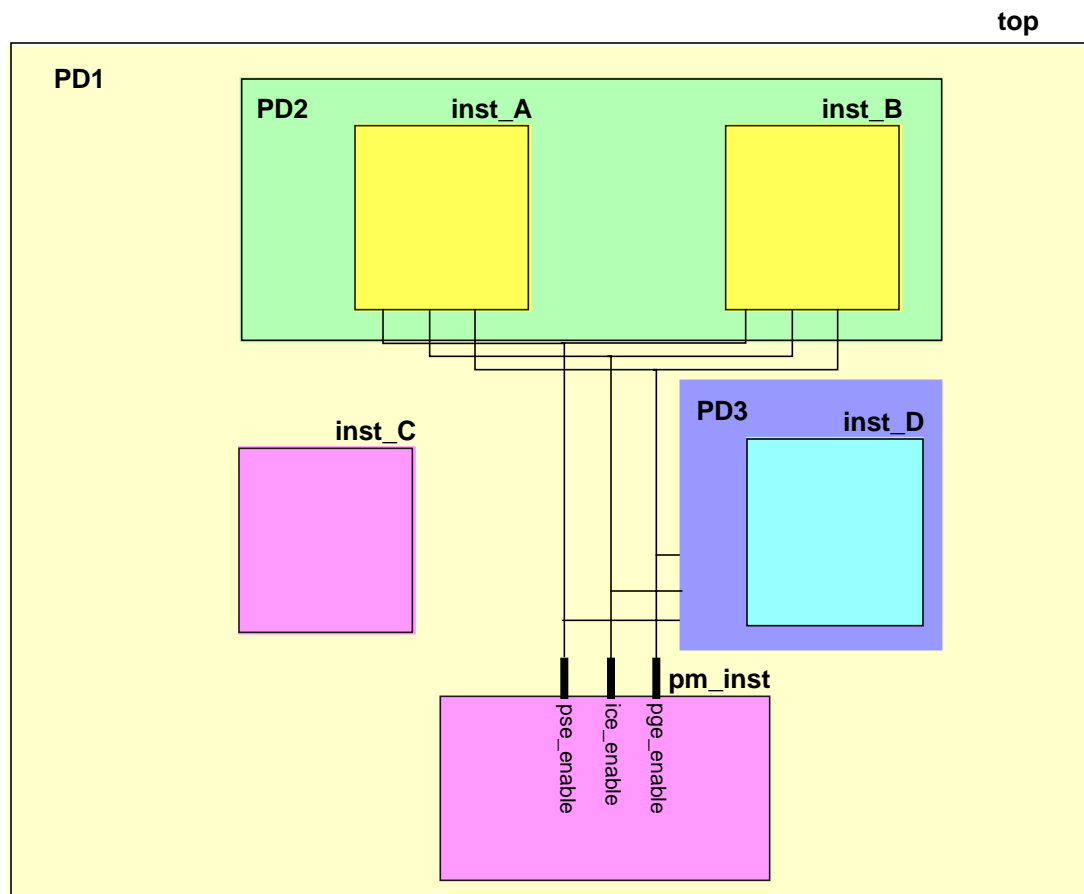
Overview

A design using power shut off (PSO) implementation is a design of which some portions can be switched on and off as needed to save leakage and dynamic power.

Logic blocks (hierarchical instances), leaf instances, and pins that use the same *main* power supply and that can be simultaneously switched on or off are said to belong to the same *power domain*. The example design in Figure 11-1 has three power domains:

- The top-level of the design, `top`, and hierarchical instances, `inst_C` and `pm_inst`, are always switched on: they belong to domain PD1
- Hierarchical instances `inst_A` and `inst_B` are always switched on and off simultaneously: they belong to power domain PD2
- Hierarchical instance `inst_D` can be switched on and off independently from hierarchical instances `inst_A` and `inst_B`: it belongs to power domain PD3

Figure 11-1 Example of Design with PSO



Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

A steady state of the design in which some power domains are switched on and some power domains are switched off is called a *power mode*. In a power mode, each power domain operates on a specific nominal condition. Table 11-1 shows the three power modes of the example design.

Table 11-1 Power Modes

Power Mode	Power Domain		
	PD1	PD2	PD3
PM1	1.1V	1.1V	1.1V
PM2	1.1V	0.0V	1.1V
PM3	1.1V	0.0V	0.0V

Note: A voltage of 0.0V indicates that the power domain is off.

To prevent that unknown states are propagated from a power domain that is powered-down to a power domain that remains on, *isolation cells* are needed at the boundaries of the power domains that are powered down. Most of the time, isolation cells are inserted at the output boundaries of the powered down domains. You can, however, also insert isolation cells at the input boundaries.

To facilitate powered down blocks to resume normal operation, *state retention cells* can be used for some sequential cells to keep their previous state prior to power down.

To connect and disconnect the power supply from the gates in a power domain, you must add *power switch logic* or use an external power shut-off method.

Special control signals are used to shut down a power domain, enable state retention, and control the working of the power switch logic and enable isolation. Table 11-2 shows the signals used in this example.

Table 11-2 Signals Controlling the Power Domains

Power Domain	Control Signals		
	power switch	isolation cell	state retention cell
PD1	no control signal	no control signal	no control signal
PD2	ps_enable[0]	ice_enable[0]	pge_enable[0]
PD3	ps_enable[1]	ice_enable[1]	pge_enable[1]

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

The Common Power Format (CPF) file captures the information described above in a text file. The [CPF File for PSO Design](#) shows how this information is described.

[PSO Information in the Design Information Hierarchy](#) shows where the additional MSV-specific information is stored.

[Flow Steps](#) shows where in the flow you need to read and execute the CPF file.

For more information on the PSO flow when you are also using DFT techniques refer to [PSO with DFT Flow](#) in *Design for Test in Encounter RTL Compiler*.

CPF File for PSO Design

The CPF file for the example in [Figure 11-1](#) on page 234 is shown in [Figure 11-2](#). [Creating a CPF File for a Design Using PSO Methodology](#) in the *Common Power Format User Guide* shows how to create a CPF file for a PSO design.

The comments in the example below show the type of information that is specified.

Figure 11-2 CPF File for PSO Example

```
#####
#           Technology part of the CPF
#####
# group libraries characterized for same operating condition
define_library_set -name set1 -libraries {lib1 lib2}
# define the isolation cells
define_isolation_cell -cells ISOLN* -enable EN -valid_location to
# define the always on cell
define_always_on_cell -cells "BUFGX2M BUFGX8M INVGX2M INVGX8M"
# define the power switch cells -- ignored by the RC-LP engine
define_power_switch_cell -cells "hd8DM hd16DM hd32DM hd64DM" \
    -stage_1_enable SLEEP -type header
define_power_switch_cell -cells "hd8M hd16M hd32M hd64M" \
    -stage_1_enable !SLEEP -type header
define_power_switch_cell -cells "ft8DM ft16DM" \
    -stage_1_enable !SLEEPN -type footer
define_power_switch_cell -cells "ft8M ft16M" \
    -stage_1_enable SLEEPN -type footer
# define the state retention cell
define_state_retention_cell -cells *DRFF* -restore_function RETN
# identify the design for which the CPF file is created
set_design top
```

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

```
# identify portions of the design that are switched on or off simultaneously
create_power_domain -name PD1 -default
create_power_domain -name PD2 -instances {inst_A inst_B} \
-shutoff_condition {pse_enable[0]}
create_power_domain -name PD3 -instances inst_D -shutoff_condition {pse_enable[1]}
# specify the operating voltages used in the design
create_nominal_condition -name off -voltage 0
create_nominal_condition -name on -voltage 1.1
# associate the nominal conditions with the power domains
create_power_mode -name PM1 -domain_conditions {PD1@on PD2@on PD3@on} -default
create_power_mode -name PM2 -domain_conditions {PD1@on PD3@on}
create_power_mode -name PM3 -domain_conditions {PD1@on}
# specify which library set to use for a specific nominal condition
update_nominal_condition -name on -library_set set1
# create rules for isolation logic insertion
create_isolation_rule -name iso1 -from PD2 \
-isolation_condition {pm_inst.ice_enable[0]}
create_isolation_rule -name iso2 -to PD1 \
-isolation_condition {pm_inst.ice_enable[1]} -isolation_output high
# create rules for state retention insertion
create_state_retention_rule -name st1 -domain PD2 \
-restore_edge {!pm_inst.pge_enable[0]}
create_state_retention_rule -name st2 -domain PD3 \
-restore_edge {!pm_inst.pge_enable[1]}
# declare power and ground nets -- ignored by the RC-LP engine
create_power_nets -nets VDD -voltage 1.1
create_power_nets -nets VDDG -voltage 0.8
create_ground_nets -nets VSS -voltage 0
# create rules for power switch insertion -- ignored by the RC-LP engine
create_power_switch_rule -name SW1 -domain PD2 -external_power_net VDD
create_power_switch_rule -name SW2 -domain PD3 -external_power_net VDD
# specify the targets for leakage and dynamic power
set_power_target -leakage 30 -dynamic 250
# specify the timing constraints and activity information
update_power_mode -name PM1 -sdc_files pm1.sdc
update_power_mode -name PM2 -sdc_files pm2.sdc
update_power_mode -name PM1 -activity_file top.tcf -activity_file_weight 100
# create global connections -- ignored by the RC-LP engine
create_global_connection -net VDD -pins VDD
create_global_connection -net VDDG -pins VDDG
create_global_connection -net VSS -pins VSS
```

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

update the rules

```
update_isolation_rules -names isol -location to -cells ISOLNX2M
```

```
update_isolation_rules -names iso2 -location to -cells ISOLNX2M
```

```
update_power_switch_rule -name SW1 -cells hd32M -prefix CDN_
```

```
update_power_switch_rule -name SW2 -cells hd32M -prefix CDN_
```

add implementation info for power domains

```
update_power_domain -name PD1 -internal_power_net VDD
```

```
update_power_domain -name PD2 -internal_power_net VDD
```

```
update_power_domain -name PD3 -internal_power_net VDD
```

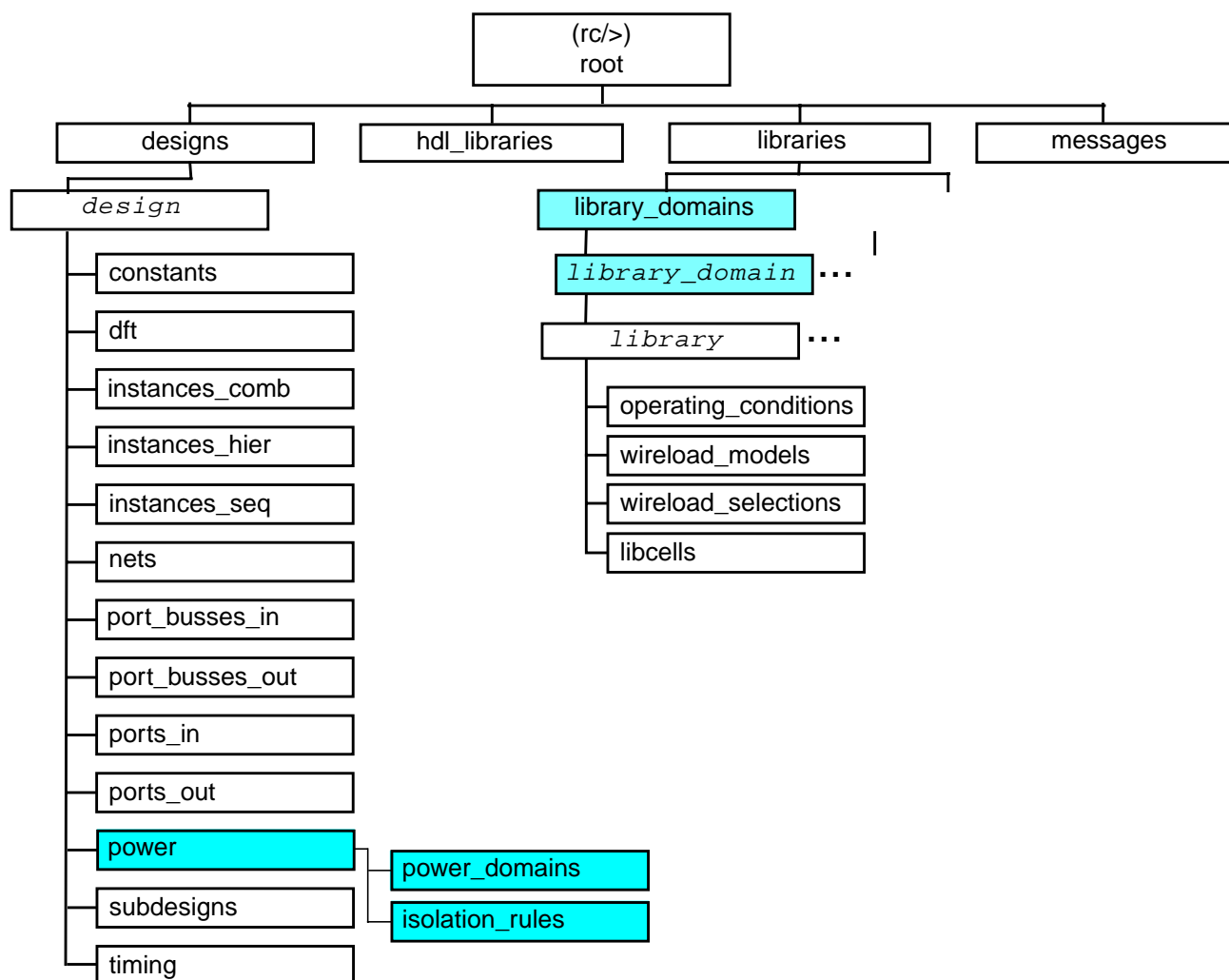
indicate when the power information for the design ends

```
end_design
```

PSO Information in the Design Information Hierarchy

RTL Compiler stores the original design data along with additional information in the CPF file in the design information hierarchy in the form of attributes. Figure 11-3 highlights the MSV information in the design information hierarchy.

Figure 11-3 Design Information Hierarchy



This section uses the comments in the CPF file shown in Figure 11-2 on page 236 to show how the information is stored in the design hierarchy.

group libraries characterized for same operating condition

The RC-LP engine creates a `library_domains` directory under the `libraries` directory. For each *library set* defined in the CPF file, it creates a subdirectory with that name in the

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

`library_domains` directory. The libraries in each set are stored in the corresponding library domain.

define the isolation cells

RTL Compiler annotates the library cells specified through `-cells` option with the following attributes:

```
set_attribute is_isolation_cell true libcell
set_attribute isolation_cell_enable_pin true libpin
```

define the always on cell

RTL Compiler annotates the library cells specified through `-cells` option with the following attribute:

```
set_attribute is_always_on true libcell
```

define the state retention cell

RTL Compiler annotates the library cells specified through `-cells` option with the following attributes:

```
set_attribute power_gating_cell true libcell
set_attribute power_gating_pin_class string libpin
set_attribute power_gating_pin_phase {active_low | active_high | none} libpin
```

identify the design for which the CPF file is created

The RC-LP engine verifies if the CPF file is defined for the design that is currently loaded.

```
# identify portions of the design that are switched on or off simultaneously
```

The RC-LP engine creates a `power` directory under the design directory with a subdirectory named after each *power domain* that was defined in the CPF file.

For any instance associated with a specific power domain, the RC-LP engine sets the following instance attribute:

```
set_attribute power_domain power_domain instance
# specify the operating voltages used in the design
# associate the nominal conditions with the power domains
# specify which library set to use for a specific nominal condition
```

The information in these three steps has this effect:

- Associates a library set with a power domain, through the following attribute setting:

```
set_attribute library_domain power_domain library_set
```

- Sets the `voltage_operating_condition` attribute to the specified value.

```
set_attribute voltage voltage library_set/library/operating_conditions/
library
```


Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

create rules for isolation logic insertion

The RC-LP engine creates a `isolation_rules` directory under the `power` directory and with a subdirectory named after each *isolation rule* defined in the CPF file.

create rules for state retention insertion

The RC-LP engine marks the instances that are candidates to be swapped with state retention cells.

specify the targets for leakage and dynamic power

When you specify these targets, the RC-LP engine sets the following design attributes:

```
set_attribute max_leakage_power power_constraint /designs/design
set_attribute max_dynamic_power power_constraint /designs/design
```

specify the timing constraints and activity information

The information specified in the SDC file is stored in the `timing` directory.
The information specified in a TCF or SAIF file is annotated on pins, ports, and nets.

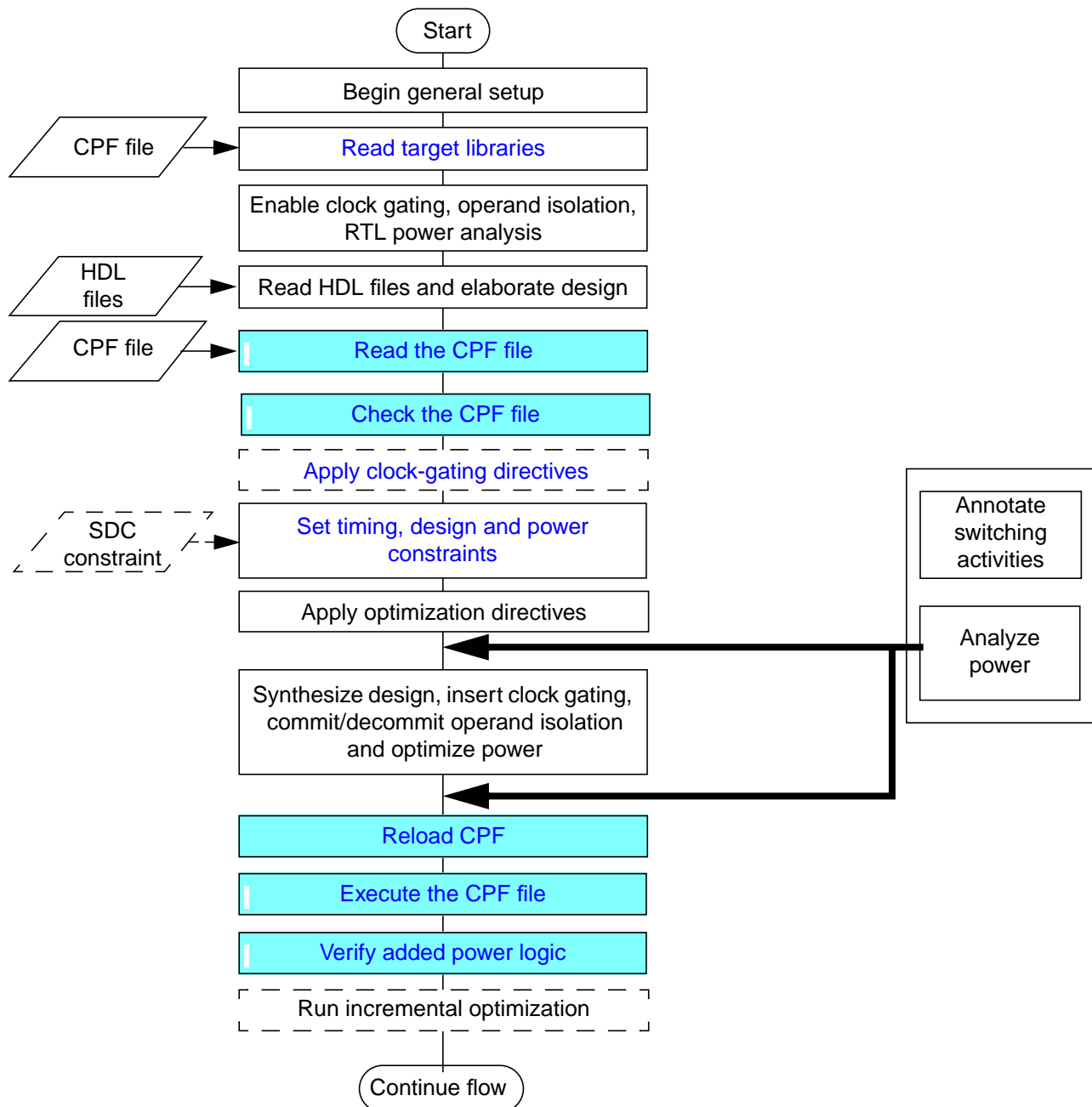
update the rules

This information is used when the CPF file is executed and isolation cells, state retention cells, and power switch logic are inserted.

Flow Steps

Figure 11-4 shows the recommended flow and highlights the tasks you need to add to the low power synthesis flow to use multiple supply voltages in the design. A script for this flow, further referred to as the *CPF-PSO* flow, is shown in [Example 11-1](#) on page 243.

Figure 11-4 Top-Down PSO Low Power Flow with CPF



Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

Example 11-1 Script for Common Top-Down CPF-PSO Flow

```
# general setup
#-----
set_attribute hdl_search_path ..

# specify the target libraries for elaboration
#-----
read_cpf -library cpf_file
check_library

# enable clock-gating, operand isolation, and RTL power analysis
#-----

set_attribute lp_insert_clock_gating true /
set_attribute lp_insert_operand_isolation true /
set_attr hdl_track_filename_row_col true /

# read and elaborate the design
#-----
read_hdl design.v
elaborate

# read the cpf file
#-----
read_cpf cpf_file
set_attribute lec_executable /tools/lec_version/bin/lec /
check_cpf

# apply clock-gating directives
#-----
set_attribute lp_clock_gating_module cg_module subdesign_list
...
set_attribute lp_clock_gating_cell libcell subdesign_list
...

# read SDC file if file is not read via the CPF file
#-----
read_sdc sd_file

# read switching activity file if file is not read via the CPF file
#-----
read_tcf tcf_file

#synthesize the design
#-----
synthesize -to_mapped

# execute the CPF file
#-----
commit_cpf
verify_power_structure
report isolation -hier -detail
report state_retention -hier -detail

#run incremental optimization
#-----
synthesize -incremental

# analyze design
#-----
report timing
report gates
report power

# export design
#-----
write_encounter design design] ...
```

Because the generic flow was described in [Chapter 2, “Recommended Flow,”](#) this chapter will focus on the additional steps and some of the steps modified for the CPF-MSV flow.

Read Target Libraries

Read the target libraries needed for elaborate the design.

```
read_cpf -library cpf_file
```

Important

The `read_cpf` command does not use the `lib_search_path` attribute setting to load libraries defined in the CPF file with the `define_library_set` command. Because the CPF file is meant to be read by multiple tools, you must provide either full or relative file name paths in the CPF file.

Tip

To specify the file paths in the CPF file, you can use environment variables.

To access Tcl variables defined in the RTL Compiler shell inside the CPF file, you need to add the global name space operator (`::`) in front of the variable. For example.

Assume the following Tcl variable is defined in your script

```
set reset_on_design 1
```

To use this variable inside your CPF file, use:

```
if {$::reset_on_design}
```

At this point the RC-LP engine creates library domains in the design information hierarchy. The names of the library domains correspond to the names of the library sets defined in the CPF file. The libraries in the library sets are stored in the corresponding library domains.

Caution

If you read in libraries to be used for elaboration only using the `library root` attribute, and you set attributes on these libraries before reading the CPF file, you may lose some of these attribute settings when reading the CPF file.

Read CPF File

Capture the power intent for your design.

```
read_cpf cpf_file
```

When reading the CPF file, the tool performs the following actions:

- Creates the power domains in the design information hierarchy.
- Stores the association of the library sets with the power domains.

You can verify the information as follows:

```
get_attribute library_domain_by_mode power_domain
```

The library set associated with the default power domain in the default power mode becomes the default library domain.

You can verify the default power mode as follows:

```
get_attribute base_mode design
```

- Stores the association of the power domains with the specified instances in the design information hierarchy.

You can verify the information as follows:

```
rc:/designs/top> get_att power_domain [find / -instance instance]
```

Important

If you marked an instance *preserved*, the power domain that the instance is associated with will still be changed. However, the instance will still be marked preserved even though it will probably be pointing to another library cell in the new power domain. In other words, the `power_domain` attribute has a higher priority than the `preserve` attribute.

If RTL Compiler cannot find the corresponding cell in the libraries that correspond to the new power domain, the instance becomes *unresolved*.

- Reads the SDC constraints file(s).
- Reads the TCF file(s).
- Marks the necessary pins and nets *preserved* to ensure technology mapping respects the power domain boundaries.
- Creates the isolation rules directories.

- Marks the instances that are candidates to be swapped with state retention cells.

Important

If you specified to use some specific library cell(s) in the `define_xxx_cell` commands but the library cells are marked `dont_use` in the `.lib` file, the RC-LP engine will use the `set_attribute preserve false libcell` and `set_attribute avoid false libcell` commands to try to mark the library cell(s) usable. To prevent this, modify the CPF file and remove those cells from the `define_xxx_cell` commands.

Check the CPF File

After reading the CPF file, check the validity of the CPF rules against the RTL of your design. This enables you to capture any violations of the low power intent of the design early in the design cycle.

`check_cpf`

Note: To run this command you need to have access to version 7.1 (or later) of the Encounter® Conformal® Low Power software.

Caution

If rule check errors are detected, you need to make the necessary changes to your CPF file before proceeding further with synthesis.

Set Timing, Design and Power Constraints

1. If you did not specify an SDC file in the CPF file, you can provide timing and design constraints at this time.

```
read_sdc sdc_file
report timing -lint
```

2. If you did not specify the targets for leakage and dynamic power optimization in the CPF file, set the following attributes:

```
set_attribute max_leakage_power float /designs/design
set_attribute max_dynamic_power float /designs/design
```

The RC-LP engine continues power optimization until the power of the design is smaller than the specified constraint. If leakage and dynamic power constraints are specified, leakage power optimization is optimized first by default.

3. To control the power optimization, use one of the following design attributes:

```
set_attribute lp_optimize_dynamic_power_first true /designs/design
set_attribute lp_power_optimization_weight float /designs/design
```

For more information on leakage power optimization, refer to [Chapter 8, “Leakage Power Optimization.”](#) For more information on dynamic power optimization, refer to [Chapter 9, “Dynamic Power Optimization.”](#)

Apply Clock-Gating and Optimization Directives

1. Select the clock-gating logic to use. You can either

- ☐ Specify which clock-gating module or integrated cell to use, by associating either of the following subdesign attributes with the appropriate subdesigns:

```
set_attribute lp_clock_gating_module cg_module subdesign_list
set_attribute lp_clock_gating_cell libcell subdesign_list
```

- ☐ Select the clock-gating cell that is available in each library domain, by setting the following design attributes (or using the default):

```
lp_clock_gating_add_obs_port
lp_clock_gating_add_reset
lp_clock_gating_control_point
lp_clock_gating_style
```

2. To control the clock gating, use the following design attributes:

```
lp_clock_gating_max_flops
lp_clock_gating_min_flops
lp_clock_gating_exclude
lp_clock_gating_extract_common_enable
```

For more information, see [Controlling Insertion of Clock-Gating Logic](#).

In addition to applying design constraints, you may need to use further optimization strategies to get the desired performance goals from synthesis.

For more information on optimization strategies and related commands, see [Defining Optimization Settings](#) in *Using Encounter RTL Compiler*.

Annotate Switching Activities

If you did not specify any activity information using the CPF file, you should annotate switching activities before power analysis for a more accurate power calculation.

For more information, see [Chapter 4, “Providing Switching Activity Information.”](#)

Estimate RTL Power

Note: This corresponds to the *Analyze Power* step before *Synthesize Design*.

RTL Power Analysis allows you to cross-reference the power consumed by the instances to the corresponding line in the RTL files. For designs using the PSO methodology, the RC-LP engine can report the average RTL power of the design

- Assuming that the entire design is always powered on (default)
- Taking into account that certain parts of the design can be shut off at certain times
- To indicate which type of power results you are interested in, set the following root attribute:

```
set_attribute lp_pso_aware_estimation {true|false} /
```

To give the correct estimation, the RC-LP engine needs to know how the switching activities annotated to the design were generated. The switching activities can have been generated

- By simulating the design assuming that the entire design is always powered on (default)
- By simulating the design while taking into account when power domains are shut off
- To indicate how the switching activities were generated, set the following design attribute:

```
set_attribute lp_pso_aware_tcf {true|false} /designs/design
```

Set these attributes before you analyze the power either using the `report power` command or using the appropriate attributes to report specific power components.

Based on the type of results and the type of switching activities provided, the RC-LP engine may need to adjust the switching activities and the power results.

Important

To perform power domain -aware power analysis, the RC-LP engine needs to know what the probability is of a power domain being shut off. This requires that you specify (user-assert) the probability of the shutoff enable signals of the power domains.

Note: The shutoff signal of a power domain is specified as follows:

```
set_attribute shutoff_signal {pin|port} power_domain
```

Four combinations are possible:

- You generated the switching activities without taking power domains into account, and you want to report the average power assuming the design is always powered on.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

Use the following settings:

```
set_attribute lp_pso_aware_estimation false /  
set_attribute lp_pso_aware_tcf false /designs/design
```

- You generated the switching activities without taking power domains into account, but you want to report the average power considering certain portions of the design can be shut off.

Use the following settings:

```
set_attribute lp_pso_aware_estimation true /  
set_attribute lp_pso_aware_tcf false /designs/design
```

You generated the switching activities taking into account when power domains were being shut off, and you want to report the considering power domains were being shut off

Use the following settings:

```
set_attribute lp_pso_aware_estimation true /  
set_attribute lp_pso_aware_tcf true /designs/design
```

- You generated the switching activities taking into account when power domains were being shut off, but you want to report the power assuming the design is always powered on.

Use the following settings:

```
set_attribute lp_pso_aware_estimation false /  
set_attribute lp_pso_aware_tcf true /designs/design
```

- To analyze the power at the RTL level, use the following command:

```
report power -rtl -detail
```

Synthesize Design

- After the constraints and optimizations are set for your design, synthesize your design using the following command:

```
synthesize -to_mapped
```

The different portions of the design that are associated with different power domains will be mapped to the target libraries of those power domains and optimized.

At this time, the selected instances are replaced with state retention cells according to the state retention rules in the CPF file.

Clock-gating insertion, commitment and decommitment of the operand isolation instances, and leakage and dynamic power optimization occur automatically during mapping and optimization.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

Note: Clock-gating insertion is power domain-aware. This implies that a clock-gating instance cannot gate flip-flops across power domains. A clock-gating instance gating the flip-flops in a power domain also belongs to that power domain.



Tip

In addition, if the setting of the `lp_clock_gating_min_flops_attribute` can not be satisfied, no clock gating will be inserted.

Reload CPF File

The netlist can change when the design is synthesized. New design objects (ports, pins, nets, and instances) can be added (for example, for DFT or for low power). To ensure the proper pin (port) selection for level shifter insertion, you need to reapply the CPF rules.

Execute CPF File

At this time the RC-LP engine inserts the isolation cells.

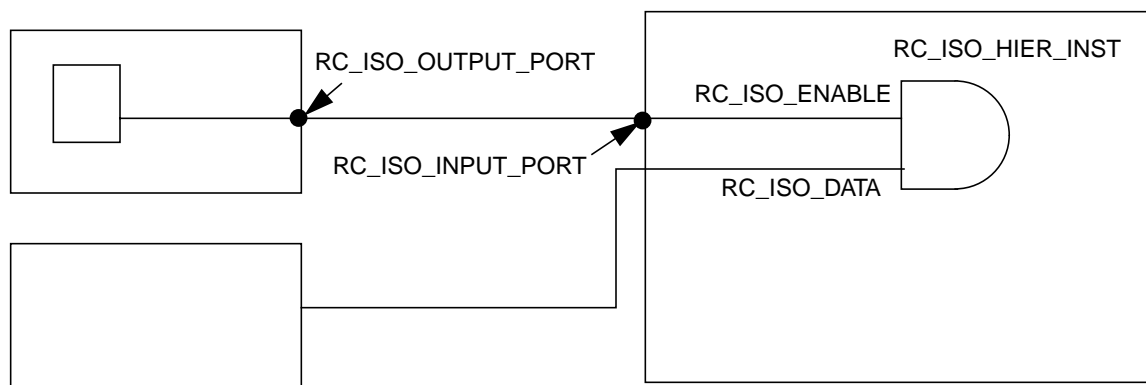
- To execute the CPF file, use the `commit_cpf` command.



Important

To connect the enable driver with the isolation enable pin, the RC-LP engine *might* need to create an input and output (sub)port. The port names are prefixed with `RC_ISO_INPUT` and `RC_ISO_OUTPUT` for input port and output port, respectively. This is shown in Figure 11-5.

Figure 11-5 Input and Output Subports Created when Connecting Enable Driver



Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

- To report all the isolation cells inserted in the design, use the `report isolation` command:

```
report isolation -hier -detail
```

Finding Isolation Logic in the Hierarchy

The RC-LP engine can group isolation cells in hierarchical instances. To be grouped into the same hierarchical instance the isolation cells must satisfy *all* the following criteria:

- They must have a common from and to power domain.
- They must have the same enable driver.
- They must have the same cell type and belong to the same library domain.

A pure isolation cell cannot be merged with a *combo* cell (level shifter and isolation).

Note: Discrete isolation cells are not considered for merging.

- They must have the same prefix.
- An isolation cell that has an inverter at its enable will only be merged with another isolation cell that also has an inverter at its enable.

You can find isolation logic-related subdesigns in the design hierarchy at

```
/designs/design/subdesigns/prefix_MOD*
```

You can find the path to an isolation logic-related hierarchical instance in the design hierarchy using either the `report isolation` command or the `find` command:

```
find / -instance prefix_HIER_INST*
```

Prefix refers to the prefix you specified using the `-prefix` option of the `update_isolation_rules` command. The default prefix is `CPF_ISO`.

Preserve Actions

The RC-LP engine automatically sets the `preserve` attribute to `size_delete_ok` on the isolation logic-related subdesigns.

```
rc:/> get_attribute preserve [find / -subdesign RC_ISO_MOD1]
size_delete_ok
```

However, if you instructed to use (a) specific library cell(s) for an isolation rule (through the `-cells` option of the `update_isolation_rules` command), the RC-LP engine sets the `preserve` attribute to `delete_ok` on the corresponding isolation hierarchy.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

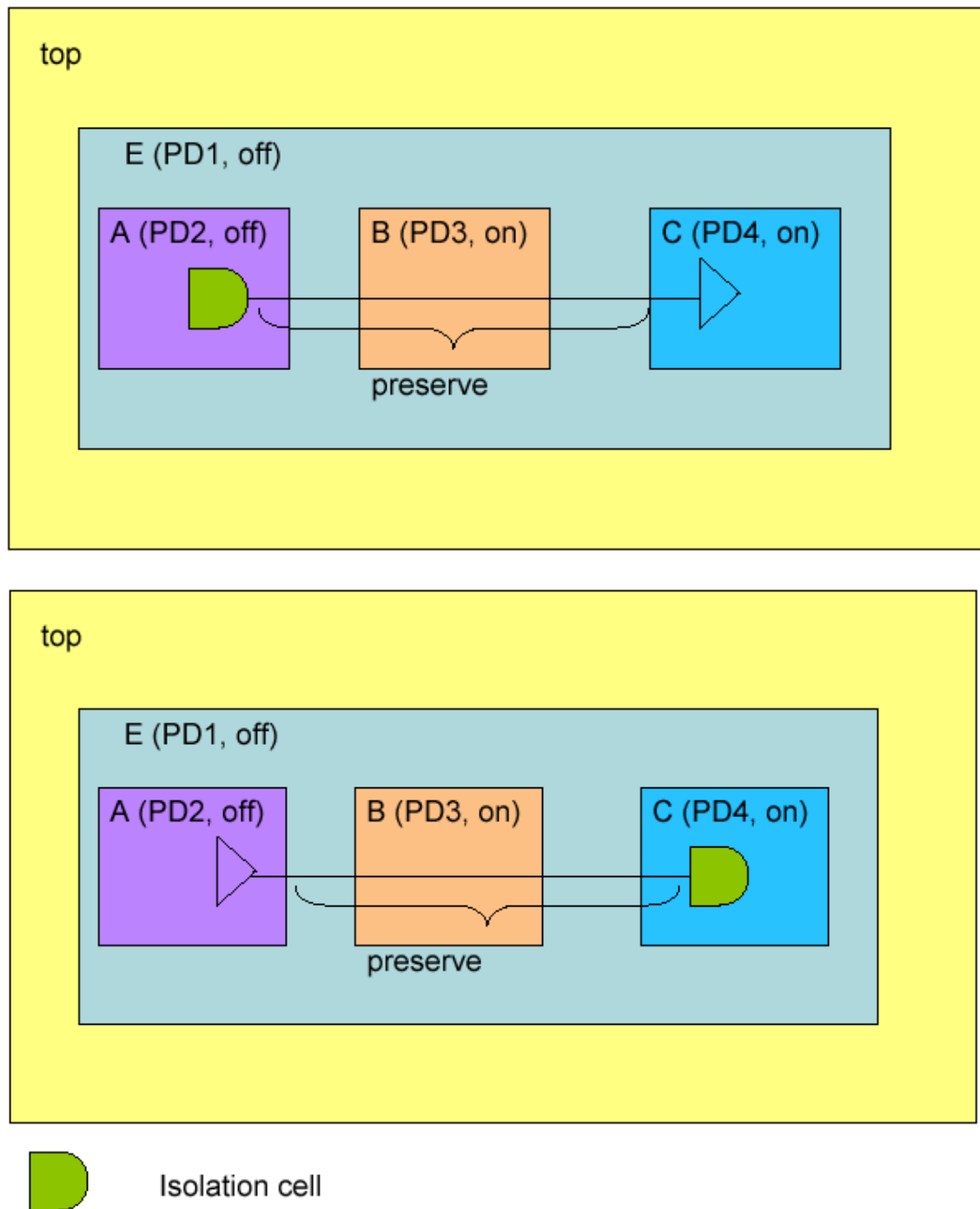
When a net crosses multiple power domains, RTL Compiler automatically sets the preserve attribute on some portions of the net to prevent buffer insertion as illustrated in [Figure 11-6](#) on page 253.

The RC-LP engine automatically sets the `ideal_driver` attribute to `true` on the enable drivers of isolation cells.

A net will be preserved if

- on the driver (input) side of the isolation cell it crosses a power domain which is different from the originating (or from) power domain
- on the load (output) side of the isolation cell, it crosses a power domain which is different from the destination (or to) power domain

Figure 11-6 Preserving Nets to Prevent Buffer Insertion



Verify Added Power Logic

Verifies whether the level shifter that were inserted in the design conform to the level shifter rules in the loaded CPF file. The tool will flag if there are any missing level shifter cells or if the level shifter cells are not connected appropriately.

```
verify_power_structure -lvl
```

Note: To run this command you need to have access to version 7.1 (or later) of the Encounter[®] Conformal[®] Low Power software.

Run Incremental Optimization

Inserting the level shifters can have an impact on the timing.

- To fix any timing issues, run incremental optimization:

```
synthesize -incremental
```

Annotate switching activities

- To get accurate power analysis results, read in the switching activity information generated from the mapped netlist:

```
read_tcf tcf_file_for_mapped_netlist
```

Analyze Power

For designs using the PSO methodology, the RC-LP engine can report the average power of the design

- Assuming that the entire design is always powered on (default)
- Taking into account that certain parts of the design can be shut off at certain times
- To indicate which type of power results you are interested in, set the following root attribute:

```
set_attribute lp_pso_aware_estimation {true|false} /
```

To give the correct estimation, the RC-LP engine needs to know how the switching activities annotated to the design were generated. The switching activities can have been generated

- By simulating the design assuming that the entire design is always powered on (default)
- By simulating the design while taking into account when power domains are shut off

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

- To indicate how the switching activities were generated, set the following design attribute:

```
set_attribute lp_pso_aware_tcf {true|false} /designs/design
```

Set these attributes before you analyze the power either using the `report power` command or using the appropriate attributes to report specific power components.

Based on the type of results and the type of switching activities provided, the RC-LP engine may need to adjust the switching activities and the power results.

Important

To perform power domain -aware power analysis, the RC-LP engine needs to know what the probability is of a power domain being shut off. This requires that you specify (user-assert) the probability of the shutoff enable signals of the power domains.

Note: The shutoff signal of a power domain is specified as follows:

```
set_attribute shutoff_signal {pin|port} power_domain
```

Four combinations are possible:

- You generated the switching activities without taking power domains into account, and you want to report the average power assuming the design is always powered on.

Use the following settings:

```
set_attribute lp_pso_aware_estimation false /  
set_attribute lp_pso_aware_tcf false /designs/design
```

- You generated the switching activities without taking power domains into account, but you want to report the average power considering certain portions of the design can be shut off.

Use the following settings:

```
set_attribute lp_pso_aware_estimation true /  
set_attribute lp_pso_aware_tcf false /designs/design
```

You generated the switching activities taking into account when power domains were being shut off, and you want to report the considering power domains were being shut off

Use the following settings:

```
set_attribute lp_pso_aware_estimation true /  
set_attribute lp_pso_aware_tcf true /designs/design
```

- You generated the switching activities taking into account when power domains were being shut off, but you want to report the power assuming the design is always powered on.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Power Shutoff Methodology

Use the following settings:

```
set_attribute lp_pso_aware_estimation false /  
set_attribute lp_pso_aware_tcf true /designs/design
```

For more information, refer to [Chapter 5, “Power Analysis.”](#)

Analyze Design

After synthesizing the design, you can generate detailed timing and area reports using the various `report` commands. For more information, see [Analyze Design](#) in the [Recommended Flow](#).

For more information on generating reports for analysis, see [Generating Reports](#) in *Using Encounter RTL Compiler* and [“Analysis Commands”](#) in the *Command Reference for Encounter RTL Compiler*.

Most reports reflect information for the library domains (sets in CPF).

Export to Place and Route

- To export the necessary files for the Encounter place and route tool, use the following command:

```
write_encounter design [design] ....
```

This command writes the netlist file, the SDC constraint file, templates for the Encounter config file, Encounter setup file, and Encounter level-shifter table file. The directory and base name of the files are controlled through the `-basename` option of the `write_encounter` command. The default base name is `rc_enc_des`.

Important

This command creates *template* files. It is your responsibility to fill in the remaining information needed to start the Encounter place and route tool.

Encounter Tcl Script Template

The `write_encounter` command creates a setup template, `rc.enc_setup`. This file sets the following information (for MSV):

1. Loads in the Encounter configuration template that it created (see **1** in Figure 11-7).
2. Loads in the original CPF file (see **2** in Figure 11-7) and commits the CPF file.
3. Loads in an MSV-specific script —this file is empty for the CPF-flow.

Figure 11-7 Extract of a Sample Encounter Setup Template File

```
#####  
#  
# First Encounter setup file  
# Created by RTL Compiler (RC) on date time  
#  
#####  
  
...  
# Design Import  
#####  
...  
loadConfig rc_enc_des/rc.conf ← 1  
  
# Mode Setup  
#####  
source rc_enc_des/rc.mode  
  
# MSV Setup  
#####  
loadCPF /home/ria/rc_ex/cpf/ls_insert/my.cpf ← 2  
commitCPF  
source rc_enc_des/rc.msv.tcl  
....
```

Encounter Config Template

The `write_encounter` command creates a template for the Encounter config file, `rc.conf`.

The following information is filled in the config file (as shown in Figure 11-8):

- The name of the netlist file (1)
- The type of the netlist file (2)
- The names of the timing libraries (3)
- The name of the constraint file (4)
- The list of the LEF files specified through the `-lef` option or read in during the RTL Compiler session (5)
- The power net names that RTL Compiler created based on the domain names (6). The power and net names are also used in the command file.

Figure 11-8 Sample Encounter Config Template File

```
#####
#
# First Encounter input configuration file
# Created by RTL Compiler (RC) on date time
#
#####
global rda_Input
set cwd lpwd
set rda_Input(ui_netlist) {rc_enc_des/rc.v} ← 1
set rda_Input(ui_netlisttype) {Verilog} ← 2
set rda_Input(ui_settop) {1}
set rda_Input(ui_topcell) {top}
set rda_Input(ui_timelib) {/my_path/lib1_1v08.lib \ ← 3
...}
set rda_Input(ui_timingcon_file) {rc_enc_des/rc.sdc} ← 4
set rda_Input(ui_buf_footprint) {BUFX2MTH}
set rda_Input(ui_inv_footprint) {INVX12MTH}
set rda_Input(ui_leffile) {} ← 5
set rda_Input(ui_cts_cell_list) {CLKBUFX8MTH ...}
set rda_Input(ui_core_cntl) {aspect}
set rda_Input(ui_aspect_ratio) {1.0000}
set rda_Input(ui_captbl_file) {}
set rda_Input(ui_defcap_scale) {1.0}
set rda_Input(ui_res_scale) {1.0}
set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_pwrnet) {VDD_PD1 VDD_PD2 ....} ← 6
set rda_Input(ui_gen_footprint) {1}

##### RTL #####
#set rda_Input(ui_netlisttype) {RTL}
```

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

- [Overview](#) on page 260
- [CPF File for DVFS Design](#)
- [Additional Information in the Design Information Hierarchy](#) on page 269
- [Flow Steps](#) on page 270
 - ❑ [Read Target Libraries](#) on page 272
 - ❑ [Read CPF File](#) on page 273
 - ❑ [Check the CPF File](#) on page 275
 - ❑ [Set Timing, Design and Power Constraints](#) on page 275
 - ❑ [Apply Clock-Gating and Optimization Directives](#) on page 276
 - ❑ [Annotate Switching Activities](#) on page 276
 - ❑ [Estimate RTL Power](#) on page 276
 - ❑ [Synthesize Design](#) on page 277
 - ❑ [Reload CPF File](#) on page 277
 - ❑ [Execute CPF File](#) on page 278
 - ❑ [Verify Added Power Logic](#) on page 282
 - ❑ [Run Incremental Optimization](#) on page 282
 - ❑ [Annotate switching activities](#) on page 282
 - ❑ [Analyze Power](#) on page 282
 - ❑ [Analyze Design](#) on page 283
 - ❑ [Export to Place and Route](#) on page 284

Overview

Dynamic voltage frequency scaling (DVFS) reduces the power in the chip by scaling down the voltage and frequency when peak performance is not required. A design using DVFS can be seen as a special case of an MSV design operating in multiple design modes.

- In a pure MSV design different portions of the design operate on different voltages and these portions *remain* operating at their respective operating voltage.
- In a DVFS design, in addition some portions can dynamically *change* to other voltages depending on the design mode or can even be switched off.

Consequently, a DVFS design must satisfy different constraints in different design modes.

DVFS designs require variable power supply(ies) that can generate the required voltage levels with minimal transition energy losses and a quick voltage transient response.

When scaling the voltage, the frequency must be scaled accordingly to meet signal propagation delay requirements.

A power scheduler can intelligently compute the appropriate frequency and voltage levels needed to execute the various applications.

Example

The `dtmf_recvr_core` design shown in [Figure 12-1](#) on page 261 is an example of a design that uses DVFS and PSO methodology. It has several other blocks which for the sake of simplicity are not shown here. However, they all operate at the same voltage as the top-level of the design.

The `dtmf_recvr_core` design further contains the following power domains.

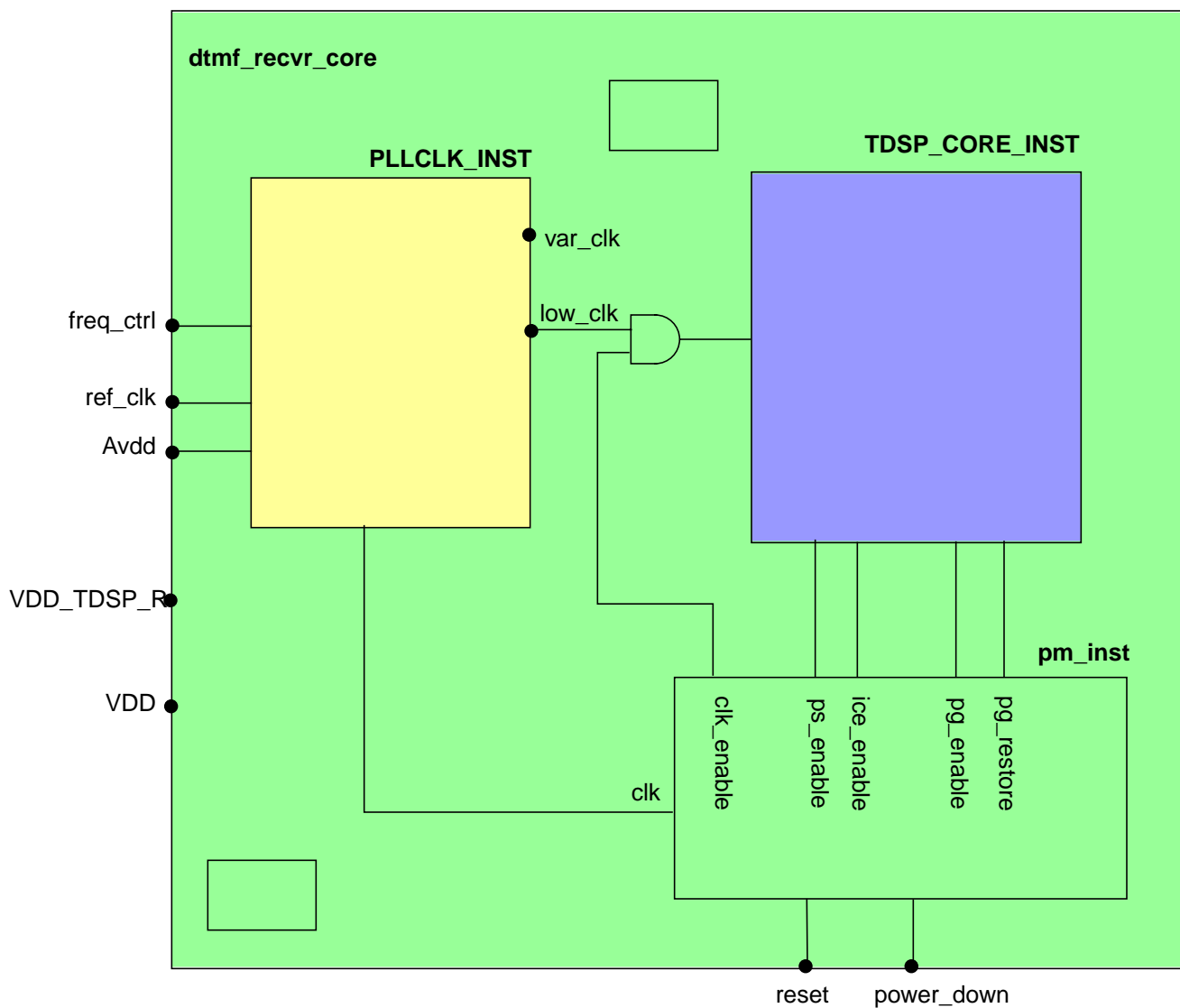
- The `PLLCLK_INST` block is the only always-on block in the design that operates at constant voltage 0.99V. This block belongs to power domain `PLL`.
- The `TDSP_CORE_INST` block operates at voltage 0.792V and it is the only block that is shut down at certain times. This block belongs to power domain `TDSPCORE`.
- The `pm_inst` block, the top-level design and the remaining blocks are always powered on but their operating voltage and frequency can change. They belong to power domain `AO`.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

Figure 12-1 Example of DVFS Design

Instances Operating on same Operating Voltage	Corresponding Power Domain	Libraries used for all modes
dtmf_recvr_core, pm_inst	AO	ao_bc_0v99, ao_wc_0v99 ao_bc_0v792, ao_wc_0v792
TDSP_CORE_INST	TDSPCORE	tdsp_bc_0v792, tdsp_wc_0v792
PLLCLK_INST	PLL	ao_bc_0v99, ao_wc_0v99



Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

A steady state of a design in which some power domains are switched on and some power domains are switched off is called a *power mode*. In a power mode, each power domain operates on a specific voltage (nominal condition). Table 12-1 shows the operating voltages for each of the power domains in the three power modes of the `dtmf_recvr_core` design. The voltages shown in this table correspond to the worst case voltages. The typical voltages for the design would be 1.1V and 0.88V.

Table 12-1 Power Modes

Power Mode	Corresponding Power Domain		
	AO	PLL	TDSPCore
full	0.99	0.99	0.792
standby	0.99	0.99	0.0
sleep	0.792	0.99	0.0

To pass signals between portions of the design that operate on different voltages, *level shifters* are needed.

To prevent unknown states from propagating from a power domain that is powered-down to a power domain that remains on, *isolation cells* are needed at the boundaries of the power domains that are powered down.

To facilitate powered down blocks to resume normal operation, *state retention cells* can be used for some sequential cells to keep their previous state prior to power up.

To connect and disconnect the power supply from the gates in a power domain, you must add *power switch logic* or use an external power shut-off method. This information is ignored by RTL Compiler.

Special control signals are used to shut down a power domain, enable state retention, restore the state of the registers when powering up a power domain, and control the working of the power switch logic. Table 12-2 shows the signals used in this design example.

Table 12-2 Signals Controlling the Power Domains

Power Domain	Control Signals		
	power switch	isolation cell	state retention cell
AO	no control signal	no control signal	no control signal
PLL	no control signal	no control signal	no control signal
TDSPCore	ps_enable	iso_enable	pg_enable and pg_restore

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

The Common Power Format (CPF) file captures the information described above in a text file. The [CPF File for DVFS Design](#) shows how this information is described.

[Additional Information in the Design Information Hierarchy](#) shows where the additional low power-specific information is stored.

Refer to [Creating a CPF File for a Design Using DVFS Methodology](#) in *Common Power Format User Guide* for more information on how to create the CPF file for this design.

[Flow Steps](#) shows where in the flow you need to read and execute the CPF file.

CPF File for DVFS Design

The CPF file for the example in [Figure 12-1](#) on page 261 is shown in [Figure 12-2](#). [Creating a CPF File for an MSV Design](#) in the *Common Power Format User Guide* shows how to create a CPF file for an MSV design.

The comments in the example below show the type of information that is specified.

Figure 12-2 CPF File for DVFS Example

```
set_cpf_version 1.0
#####
#           Technology part of the CPF
#####

set libdir ../LIBS
set lib_0v99_wc " $libdir/timing/tcbn45lpbwp_c060907wc.lib "
set lib_ao_wc " $libdir/timing/tcbn45lpbwp_wc0d720d9.lib \
                $libdir/timing/pllclk_slow.lib \
                $libdir/timing/ram_256x16A_slow.lib \
                $libdir/timing/rom_512x16A_slow.lib "
set lib_0v99_bc " $libdir/N45/timing/tcbn45lpbwp_c060907bc.lib "
set lib_ao_bc " $libdir/timing/tcbn45lpbwp_bc0d88ld1.lib \
                $libdir/timing/pllclk_slow.lib \
                $libdir/timing/ram_256x16A_slow.lib \
                $libdir/timing/rom_512x16A_slow.lib "
set lib_0v792_wc " $libdir/timing/tcbn45lpbwp_c060907wc0d72.lib "
set lib_tdsp_wc " $libdir/timing/tcbn45lpbwp_wc0d90d72.lib \
                 $libdir/timing/tcbn45lpbwphvt_wc0d72.lib \
                 $libdir/timing/tcbn45lpbwp_wc0d72_ptlv1.lib "
set lib_0v792_bc " $libdir/timing/tcbn45lpbwp_c060907bc0d88.lib "
set lib_tdsp_bc " $libdir/timing/tcbn45lpbwp_bclld10d88.lib
                 $libdir/timing/tcbn45lpbwphvt_bc0d88.lib \
                 $libdir/timing/tcbn45lpbwp_bc0d88_ptlv1.lib "

# define the library sets
define_library_set -name ao_wc_0v99 -libraries "$lib_0v99_wc $lib_ao_wc"
define_library_set -name ao_bc_0v99 -libraries "$lib_0v99_bc $lib_ao_bc"
define_library_set -name ao_wc_0v792 -libraries "$lib_0v792_wc $lib_ao_wc"
define_library_set -name ao_bc_0v792 -libraries "$lib_0v792_bc $lib_ao_bc"
define_library_set -name tdsp_wc_0v792 -libraries "$lib_0v792_wc $lib_tdsp_wc"
define_library_set -name tdsp_bc_0v792 -libraries "$lib_0v792_bc $lib_tdsp_bc"

# define the level shifters
define_level_shifter_cell -cells LVL*HLD* \
    -input_voltage_range 0.792:0.99:0.099 \
    -output_voltage_range 0.792:0.99:0.099 \
    -direction down \
    -output_power_pin VDD \
```


Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

```
-ground VSS \
-valid_location to
define_level_shifter_cell -cells PTLVL*HLD* \
-input_voltage_range 0.792:0.99:0.099 \
-output_voltage_range 0.792:0.99:0.099 \
-direction down \
-output_power_pin TVDD \
-ground VSS \
-valid_location to
define_level_shifter_cell -cells LVLLHCD* \
-input_voltage_range 0.792:0.99:0.099 \
-output_voltage_range 0.792:0.99:0.099 \
-output_voltage_input_pin NSLEEP \
-input_power_pin VDDL \
-output_power_pin VDD \
-direction up \
-ground VSS \
-valid_location to
define_level_shifter_cell -cells LVLLHD* \
-input_voltage_range 0.792:0.99:0.099 \
-output_voltage_range 0.792:0.99:0.099 \
-input_power_pin VDDL \
-output_power_pin VDD \
-direction up \
-ground VSS \
-valid_location to
# define the isolation cells
define_isolation_cell -cells LVLLHCD* \
-power VDD \
-ground VSS \
-enable NSLEEP \
-valid_location to
# define the power switch cells
define_power_switch_cell -cells {HDRDID1BWPHVT HDRDIAOND1BWPHVT} \
-power_switchable VDD -power TVDD \
-stage_1_enable !NSLEEPIN1 \
-stage_1_output NSLEEPOUT1 \
-stage_2_enable !NSLEEPIN2 \
-stage_2_output NSLEEPOUT2 \
-type header
# define the state retention cell
define_state_retention_cell -cells { RSDFCRHD2BWP } \
-clock_pin CP \
-power TVDD \
-power_switchable VDD \
-ground VSS \
-save_function "SAVE" \
-restore_function "!NRESTORE"
# define the always on cell
define_always_on_cell -cells {PTBUFFD2BWP} \
-power_switchable VDD -power TVDD -ground VSS

#####
#      Design part of the CPF
#####
```

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

```
set_design dtmf_recvr_core
set_hierarchy_separator "/"
set_constraintDir ../mmmc

# create power domains
create_power_domain -name AO -default
create_power_domain -name TDSPCore -instances TDSP_CORE_INST \
    -shutoff_condition {PM_INST/ps_enable}
create_power_domain -name PLL -instances PLLCLK_INST \
    -boundary_ports {refclk vcom vcop ibias pllrst}

# create nominal conditions
create_nominal_condition -name high_ao -voltage 0.99
create_nominal_condition -name low_ao -voltage 0.792
create_nominal_condition -name low_tdsp -voltage 0.792
create_nominal_condition -name off -voltage 0

# declare power and ground nets
create_power_nets -nets VDD -voltage {0.792:0.99:0.198}
create_power_nets -nets VDD_TDSP_R -voltage 0.792
create_power_nets -nets Avdd -voltage 0.99
create_power_nets -nets VDD_TDSPCore -internal -voltage 0.792
create_ground_nets -nets VSS
create_ground_nets -nets Avss

# create power modes
create_power_mode -name full \
    -domain_conditions {AO@high_ao PLL@high_ao TDSPCore@low_tdsp} -default
create_power_mode -name standby \
    -domain_conditions {AO@high_ao PLL@high_ao TDSPCore@off}
create_power_mode -name sleep \
    -domain_conditions {AO@low_ao PLL@high_ao TDSPCore@off}

# associate library sets with nominal conditions
update_nominal_condition -name high_ao -library_set ao_wc_0v99
update_nominal_condition -name low_ao -library_set ao_wc_0v792
update_nominal_condition -name low_tdsp -library_set tdsp_wc_0v792

# create rules for level shifter insertion
create_level_shifter_rule -name LSRULE_H2L -from AO -to TDSPCore \
    -exclude {PM_INST/ps_enable PM_INST/pg_enable PM_INST/pg_restore}
create_level_shifter_rule -name LSRULE_H2L_AO -from AO -to TDSPCore \
    -pins {PM_INST/ps_enable PM_INST/pg_enable PM_INST/pg_restore}
create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to AO

# create rule for isolation logic insertion
create_isolation_rule -name ISORULE -from TDSPCore \
    -isolation_condition "!PM_INST/iso_enable" -isolation_output high

# create rule for power switch insertion
create_power_switch_rule -name TDSPCore_SW -domain TDSPCore \
    -external_power_net VDD_TDSP_R
```

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

```
# create rule for state retention insertion
create_state_retention_rule -name SRPG_TDSP \
    -domain TDSPCore \
    -restore_edge {!PM_INST/pg_restore} \
    -save_edge {PM_INST/pg_enable}

# specify timing constraints
update_power_mode -name full \
    -sdc_files ${constraintDir}/dtmf_recvr_core_gate.sdc
update_power_mode -name standby \
    -sdc_files ${constraintDir}/dtmf_recvr_core_gate.sdc
update_power_mode -name sleep \
    -sdc_files ${constraintDir}/dtmf_recvr_core_dull.sdc
#####
# Additional Information for Physical Implementation
#####
# create operating corners
create_operating_corner -name BCCOM_AO \
    -process 1 -temperature 0 -voltage 1.21 -library_set ao_bc_0v99
create_operating_corner -name WCCOM_AO \
    -process 1 -temperature 125 -voltage 0.99 -library_set ao_wc_0v99
create_operating_corner -name BC08COM_AO \
    -process 1 -temperature 0 -voltage 0.968 -library_set ao_bc_0v792
create_operating_corner -name BC08COM_TDSP \
    -process 1 -temperature 0 -voltage 0.968 -library_set tdsp_bc_0v792
create_operating_corner -name WC08COM_AO \
    -process 1 -temperature 125 -voltage 0.792 -library_set ao_wc_0v792
create_operating_corner -name WC08COM_TDSP \
    -process 1 -temperature 125 -voltage 0.792 -library_set tdsp_wc_0v792

# create analysis views
create_analysis_view -name AV_full_MIN_RC1 -mode full \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}
create_analysis_view -name AV_full_MIN_RC2 -mode full \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}
create_analysis_view -name AV_full_MAX_RC1 -mode full \
    -domain_corners {AO@WCCOM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}
create_analysis_view -name AV_full_MAX_RC2 -mode full \
    -domain_corners {AO@WCCOM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}
create_analysis_view -name AV_standby_MIN_RC1 -mode standby \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}
create_analysis_view -name AV_standby_MAX_RC1 -mode standby \
    -domain_corners {AO@WCCOM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}
create_analysis_view -name AV_sleep_MIN_RC1 -mode sleep \
    -domain_corners {AO@BC08COM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}
create_analysis_view -name AV_sleep_MAX_RC1 -mode sleep \
    -domain_corners {AO@WC08COM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}

# create global connections
create_global_connection -domain AO -net VDD_TDSPCore -pins VDDL
create_global_connection -domain AO -net VDD -pins VDD
```

Low Power in Encounter RTL Compiler

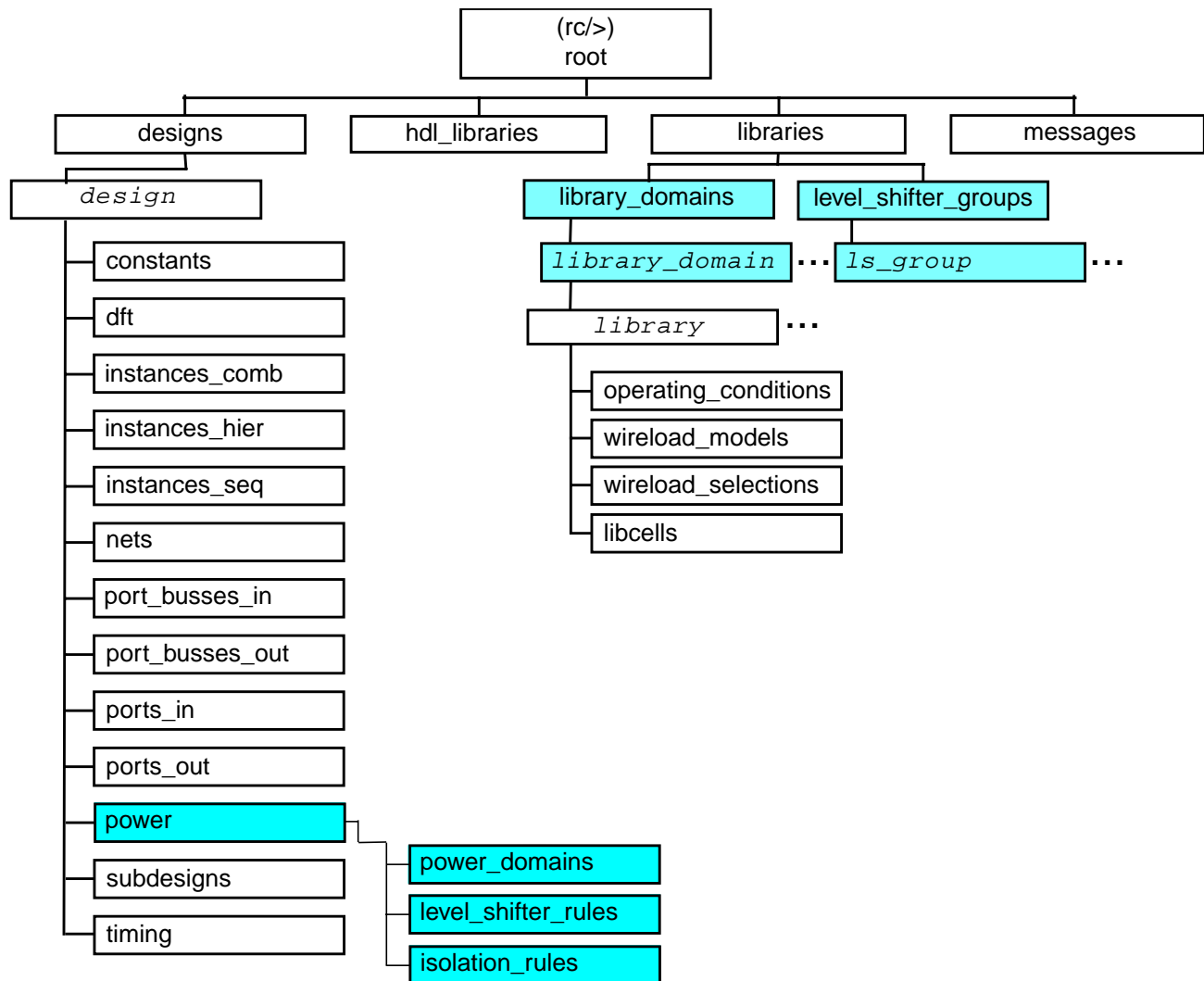
Using CPF for Designs Using Dynamic Voltage Frequency Scaling

```
create_global_connection -domain AO -net VSS -pins VSS
create_global_connection -domain PLL -net Avdd -pins avdd!
create_global_connection -domain PLL -net Avss -pins agnd!
create_global_connection -domain PLL -net VDD -pins VDDL
create_global_connection -domain PLL -net Avdd -pins VDD
create_global_connection -domain PLL -net Avss -pins VSS
create_global_connection -domain TDSPCore -net VSS -pins VSS
create_global_connection -domain TDSPCore -net VDD_TDSP_R -pins TVDD
create_global_connection -domain TDSPCore -net VDD_TDSPCore -pins VDD
# add implementation info for power domains
update_power_domain -name AO -internal_power_net VDD
update_power_domain -name TDSPCore -internal_power_net VDD_TDSPCore
update_power_domain -name PLL -internal_power_net Avdd
# update the rules
update_level_shifter_rules -names LSRULE_H2L -cells LVLHLD2BWP -location to
update_level_shifter_rules -names LSRULE_H2L_AO -cells PTLVLHLD2BWP -location to
update_level_shifter_rules -names LSRULE_H2L_PLL -cells LVLHLD2BWP -location to
update_isolation_rules -names ISORULE -location to -cells LVLLHCD2BWP
update_power_switch_rule -name TDSPCore_SW -cells HDRDID1BWPHVT \
    -prefix CDN_SW_ -acknowledge_receiver switch_en_out
update_state_retention_rules -names SRPG_TDSP \
    -cell RSDFCRHD2BWP -library_set tdsp_wc_0v792
end_design
```

Additional Information in the Design Information Hierarchy

RTL Compiler stores the original design data along with additional information in the CPF file in the design information hierarchy in the form of attributes. Figure 12-3 highlights where the information is stored in the design information hierarchy.

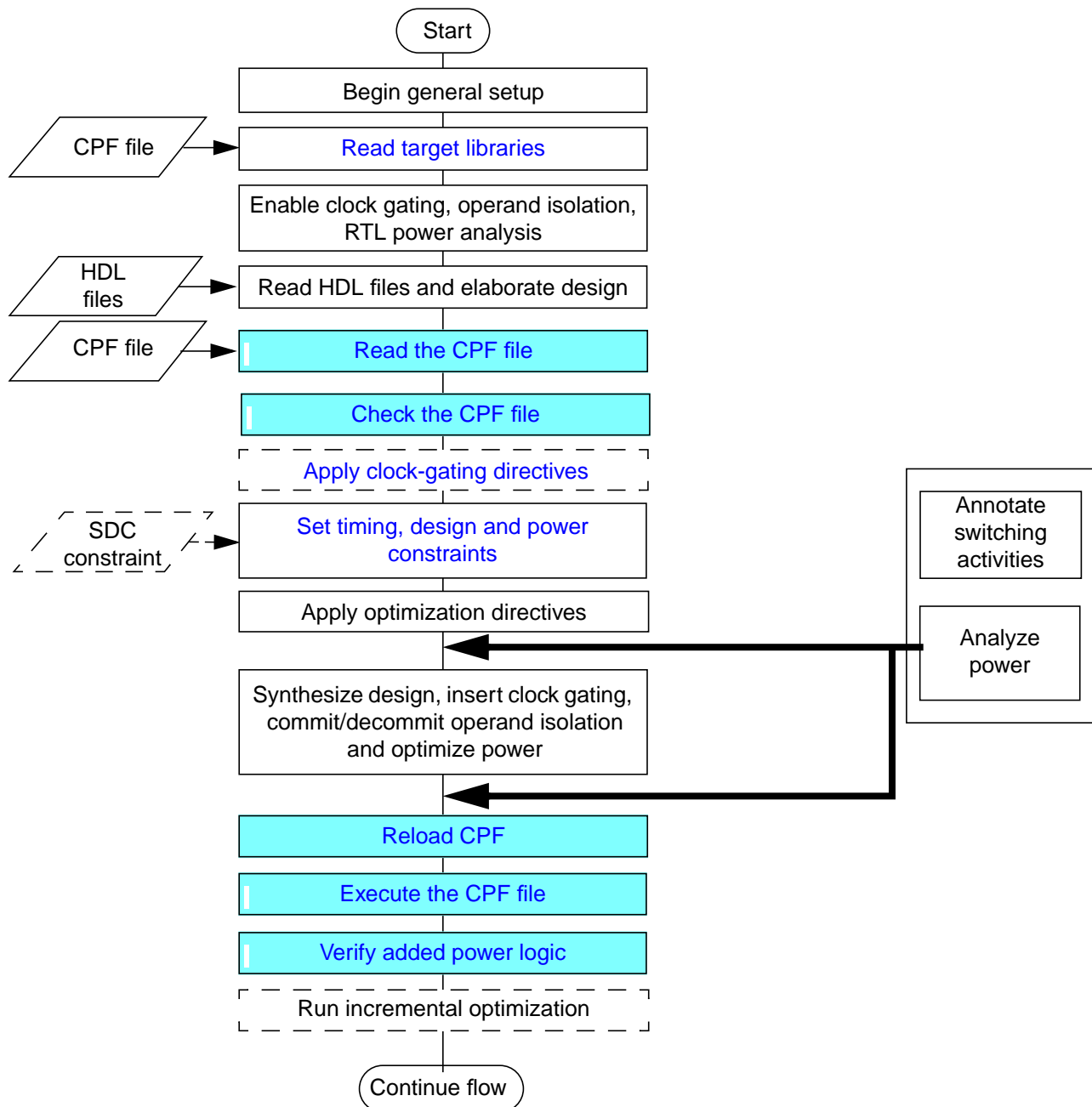
Figure 12-3 Design Information Hierarchy



Flow Steps

Figure 12-4 shows the recommended flow and highlights the tasks you need to add to the low power synthesis flow to use multiple supply voltages in the design. A script for this flow, further referred to as the *CPF-MSV* flow, is shown in [Example 12-1](#) on page 271.

Figure 12-4 Top-Down DVFS Low Power Flow with CPF



Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

Example 12-1 Script for Common Top-Down CPF-DVFS Flow

```
# general setup
#-----
set_attribute hdl_search_path ..
# specify the target libraries for elaboration
#-----
read_cpf -library cpf_file
check_library
# enable clock-gating, operand isolation, and RTL power analysis
#-----
set_attribute lp_insert_clock_gating true /
set_attribute lp_insert_operand_isolation true /
set_attr hdl_track_filename_row_col true /
# read and elaborate the design
#-----
read_hdl design.v
elaborate
# read the cpf file
#-----
read_cpf cpf_file
set_attribute lec_executable /tools/lec_version/bin/lec /
check_cpf
# apply clock-gating directives
#-----
set_attribute lp_clock_gating_module cg_module subdesign_list
...
set_attribute lp_clock_gating_cell libcell subdesign_list
...
# read SDC file if file is not read via the CPF file
#-----
read_sdc sdc_file
# read switching activity file if file is not read via the CPF file
#-----
read_tcf tcf_file
#synthesize the design
#-----
synthesize -to_mapped
# execute the CPF file
#-----
commit_cpf
report level_shifter -hier -detail
report isolation -hier -detail
report state_retention -hier -detail
verify_power_structure
#run incremental optimization
#-----
synthesize -incremental
# analyze design
#-----
report timing
...
# export design
#-----
write_encounter design [design] ...
```

Because the generic flow was described in [Chapter 2, “Recommended Flow,”](#) this chapter will focus on the additional steps and some of the steps modified for the CPF flow.

Read Target Libraries

Read the target libraries needed to elaborate the design.

For a CPF-based flow, it is recommended to read the target technology libraries defined in the CPF file.

```
read_cpf -library cpf_file
```



The `read_cpf` command does not use the `lib_search_path` attribute setting to load libraries defined in the CPF file with the `define_library_set` command. Because the CPF file is meant to be read by multiple tools, you must provide either full or relative file name paths in the CPF file.



To specify the file paths in the CPF file, you can use environment variables.

To access Tcl variables defined in the RTL Compiler shell inside the CPF file, you need to add the global name space operator (`::`) in front of the variable. For example.

Assume the following Tcl variable is defined in your script

```
set reset_on_design 1
```

To use this variable inside your CPF file, use:

```
if {$::reset_on_design}
```

At this point the RC-LP engine

- Creates library domains in the design information hierarchy.

The names of the library domains correspond to the names of the library sets defined in the CPF file. The libraries in the library sets are stored in the corresponding library domains.

- Creates the level shifter groups according to the CPF specifications

The RC-LP engine creates a level shifter group for each input and output voltage range pair in the `define_level_shifter_cell` commands to point to the library cells that are defined as level shifters in this category.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

More specifically, level shifters will be in same level shifter group if the following criteria are met:

- ❑ The level shifters belong to same library domain
- ❑ The level shifters have the same input and output voltage range
- ❑ The level shifters have the same direction, valid location and enable voltage
- ❑ The level shifters have the same functionality

For example, an AND gate can not be grouped with a buffer.

Important

If you specified to use some specific library cell(s) in the `define_xxx_cell` commands but the library cells are marked `dont_use` in the `.lib` file, the RC-LP engine will use the `set_attribute preserve false libcell` and `set_attribute avoid false libcell` commands to try to mark the library cell(s) usable. To prevent this, modify the CPF file and remove those cells from the `define_xxx_cell` commands.

Caution

If you read in libraries to be used for elaboration only using the `library root` attribute, the RC-LP engine will remove these libraries when the CPF file is read in. This implies that any attributes set on these libraries will be lost when reading in the CPF file.

Read CPF File

Capture the power intent for your design.

`read_cpf cpf_file`

When reading the CPF file, the tool performs the following actions:

- Creates the power domains in the design information hierarchy
- Stores the power mode information.

In RTL Compiler, the power modes are handled as design modes.

You can verify that a power domain is setup for DVFS synthesis by examining the

`get_attribute library_domain_by_mode power_domain`

This attribute associates the appropriate library domain (corresponding to an operating voltage) to each mode.

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling



Tip

Specify the power mode in which the design will be operating most frequently as the default power mode in the CPF file.

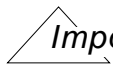
You can verify the default power mode as follows:

```
get_attribute base_mode design
```

- Stores the association of the power domains with the specified instances in the design information hierarchy.

You can verify the information as follows:

```
rc:/designs/top> get_att power_domain [find / -instance instance]
```

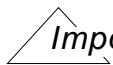


Important

If you marked an instance *preserved*, the power domain that the instance is associated with will still be changed. However, the instance will still be marked preserved even though it will probably be pointing to another library cell in the new power domain. In other words, the `power_domain` attribute has a higher priority than the `preserve` attribute.

If RTL Compiler cannot find the corresponding cell in the libraries that correspond to the new power domain, the instance becomes *unresolved*.

- Reads the SDC constraints file(s)
- Reads the TCF file(s).
- Marks the necessary pins and nets *preserved* to ensure technology mapping respects the power domain boundaries.
- Creates the level-shifter rules directories.
- Creates the isolation rules directories.
- Marks the instances that are candidates to be swapped with state retention cells.



Important

If you specified to use some specific library cell(s) in the `define_xxx_cell` commands but the library cells are marked `dont_use` in the `.lib` file, the RC-LP engine will use the `set_attribute preserve false libcell` and `set_attribute avoid false libcell` commands to try to mark the library cell(s) usable. To prevent this, modify the CPF file and remove those cells from the `define_xxx_cell` commands.

Check the CPF File

After reading the CPF file, check the validity of the CPF rules against the RTL of your design. This enables you to capture any violations of the low power intent of the design early in the design cycle.

`check_cpf`

Note: To run this command you need to have access to version 7.1 (or later) of the Encounter[®] Conformal[®] Low Power software.



Caution

If rule check errors are detected, you need to make the necessary changes to your CPF file before proceeding further with synthesis.

Set Timing, Design and Power Constraints

1. If you did not specify an SDC file in the CPF file, you can provide timing and design constraints at this time.

```
read_sdc sdc_file
report timing -lint
```

2. If you did not specify the targets for leakage and dynamic power optimization in the CPF file, set the following attributes:

```
set_attribute max_leakage_power float /designs/design
set_attribute max_dynamic_power float /designs/design
```

The RC-LP engine continues power optimization until the power of the design is smaller than the specified constraint. If leakage and dynamic power constraints are specified, leakage power optimization is optimized first by default.

3. To control the power optimization, use one of the following design attributes:

```
set_attribute lp_optimize_dynamic_power_first true /designs/design
set_attribute lp_power_optimization_weight float /designs/design
```

For more information on leakage power optimization, refer to [Chapter 8, “Leakage Power Optimization.”](#) For more information on dynamic power optimization, refer to [Chapter 9, “Dynamic Power Optimization.”](#)

Apply Clock-Gating and Optimization Directives

1. Select the clock-gating logic to use. You can either

- ❑ Specify which clock-gating module or integrated cell to use, by associating either of the following subdesign attributes with the appropriate subdesigns:

```
set_attribute lp_clock_gating_module cg_module subdesign_list
set_attribute lp_clock_gating_cell libcell subdesign_list
```

- ❑ Select the clock-gating cell that is available in each library domain, by setting the following design attributes (or using the default):

```
lp_clock_gating_add_obs_port
lp_clock_gating_add_reset
lp_clock_gating_control_point
lp_clock_gating_style
```

2. To control the clock gating, use the following design attributes:

```
lp_clock_gating_max_flops
lp_clock_gating_min_flops
lp_clock_gating_exclude
lp_clock_gating_extract_common_enable
```

For more information, see [Controlling Insertion of Clock-Gating Logic](#).

In addition to applying design constraints, you may need to use further optimization strategies to get the desired performance goals from synthesis.

For more information on optimization strategies and related commands, see [Defining Optimization Settings](#) in *Using Encounter RTL Compiler*.

Annotate Switching Activities

If you did not specify any activity information using the CPF file, you should annotate switching activities before power analysis for a more accurate power calculation.

For more information, see [Chapter 4, “Providing Switching Activity Information.”](#)

Estimate RTL Power

Note: This corresponds to the *Analyze Power* step before *Synthesize Design*.

- To analyze the power at the RTL level, use the following command:

```
report power -rtl -detail
```

For more information, refer to [Chapter 5, “Power Analysis.”](#)

Synthesize Design

- After the constraints and optimizations are set for your design, synthesize your design using the following command:

```
synthesize -to_mapped
```

The different portions of the design that are associated with different power domains will be mapped to the target libraries of those power domains and optimized.

At this time, the selected instances are replaced with state retention cells according to the state retention rules in the CPF file.

Clock-gating insertion, commitment and decommitment of the operand isolation instances, and leakage and dynamic power optimization occur automatically during mapping and optimization.

Note: Clock-gating insertion is power domain-aware. This implies that a clock-gating instance cannot gate flip-flops across power domains. A clock-gating instance gating the flip-flops in a power domain also belongs to that power domain.



Tip

In addition, if the setting of the lp_clock_gating_min_flops attribute can not be satisfied, no clock gating will be inserted.

Reload CPF File

The netlist can change when the design is synthesized. New design objects (ports, pins, nets, and instances) can be added (for example, for DFT or for low power). To ensure the proper pin (port) selection for level shifter insertion, you need to reapply the CPF rules.

Execute CPF File

At this time the RC-LP engine inserts the level shifters and isolation cells.

- To execute the CPF file, use the `commit cpf` command.

RTL Compiler can only insert level shifters or isolation cells between two power domains if you defined the appropriate rules in the CPF file.

RTL Compiler will not insert level shifters on

- ☐ constant nets
 - ☐ dangling pins or nets
 - ☐ multiple-driver nets if all or some drivers reside in different library domains and you request to insert the level shifters in the destination power domain (to domain)
- To report all the level shifters inserted in the design, use the `report level_shifter` command:

```
report level_shifter -hier -detail
```

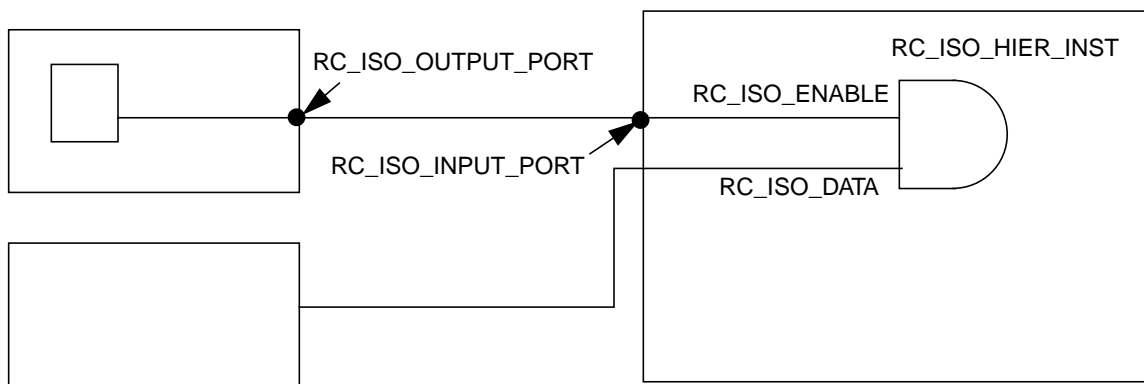
- To report all the isolation cells inserted in the design, use the `report isolation` command:

```
report isolation -hier -detail
```

Important

To connect the enable driver with the isolation enable pin, the RC-LP engine *might* need to create an input and output (sub)port. The port names are prefixed with `RC_ISO_INPUT` and `RC_ISO_OUTPUT` for input port and output port, respectively. This is shown in Figure 12-5.

Figure 12-5 Input and Output Subports Created when Connecting Enable Driver



Finding Level Shifters and Isolation Cells in the Hierarchy

RTL Compiler creates a separate subdesign for the level shifters connecting two power domains. You can find level-shifter subdesigns in the design hierarchy at

```
/designs/design/subdesigns/prefix_MOD*
```

You can find the path to a level-shifter instance in the design hierarchy using the `find` command:

```
find / -instance prefix_HIER_INST*
```

Prefix refers to the prefix you specified using the `-prefix` option of the `update_level_shifter_rules` command. The default prefix is `CPF_LS`.

You can find isolation logic-related subdesigns in the design hierarchy at

```
/designs/design/subdesigns/prefix_MOD*
```

You can find the path to an isolation logic-related hierarchical instance in the design hierarchy using either the `report isolation` command or the `find` command:

```
find / -instance prefix_HIER_INST*
```

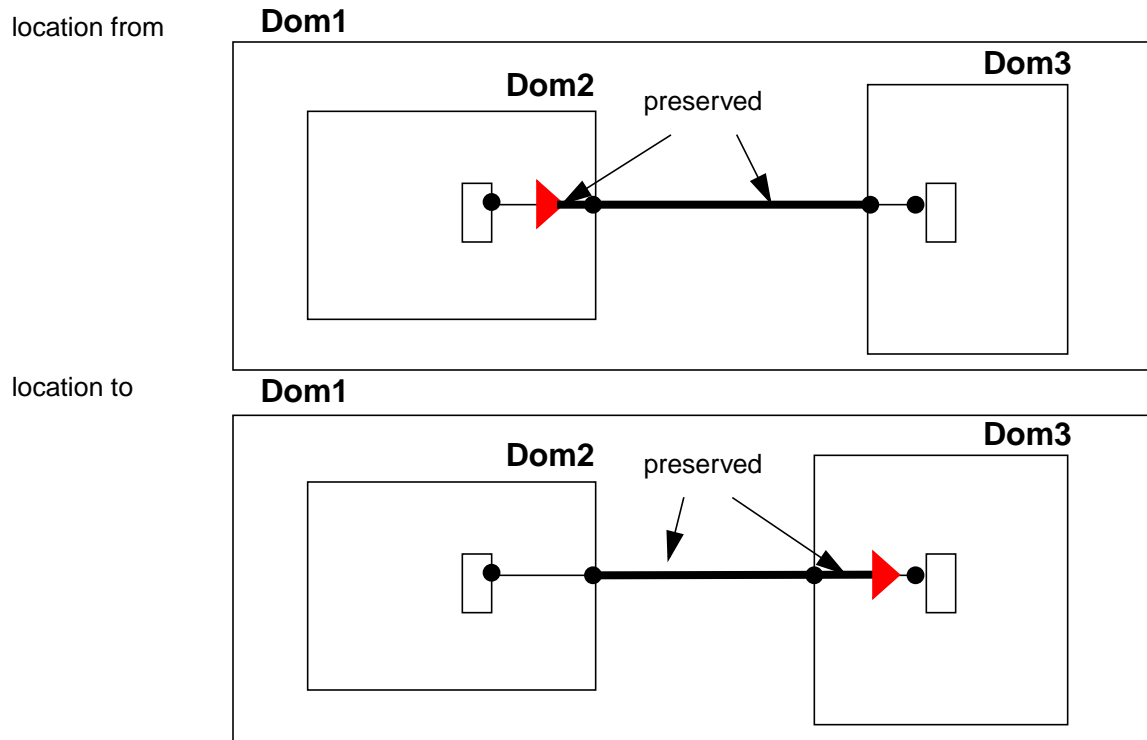
Prefix refers to the prefix you specified using the `-prefix` option of the `update_isolation_rules` command. The default prefix is `CPF_ISO`.

Preserve Actions

RTL Compiler automatically sets the `preserve` attribute on the level-shifter subdesigns to `size_delete_ok`. However, if you use a dedicated library cell for level shifter insertion, RTL Compiler sets the `preserve` attribute value to `delete_ok` on the level-shifter subdesigns.

When a net crosses multiple library domains, RTL Compiler automatically sets the `preserve` attribute on some portions of the net to prevent buffer insertion as illustrated in Figure .

Figure 12-6 Preserving Nets to Prevent Buffer Insertion



The RC-LP engine automatically sets the `preserve` attribute to `size_delete_ok` on the isolation logic-related subdesigns.

```
rc:/> get_attribute preserve [find / -subdesign RC_ISO_MOD1]
size_delete_ok
```

However, if you instructed to use (a) specific library cell(s) for an isolation rule (through the `-cells` option of the `update_isolation_rules` command), the RC-LP engine sets the `preserve` attribute to `delete_ok` on the corresponding isolation hierarchy.

When a net crosses multiple power domains, RTL Compiler automatically sets the `preserve` attribute on some portions of the net to prevent buffer insertion as illustrated in [Figure 12-7](#) on page 281.

The RC-LP engine automatically sets the `ideal_driver` attribute to `true` on the enable drivers of isolation cells.

A net will be preserved if

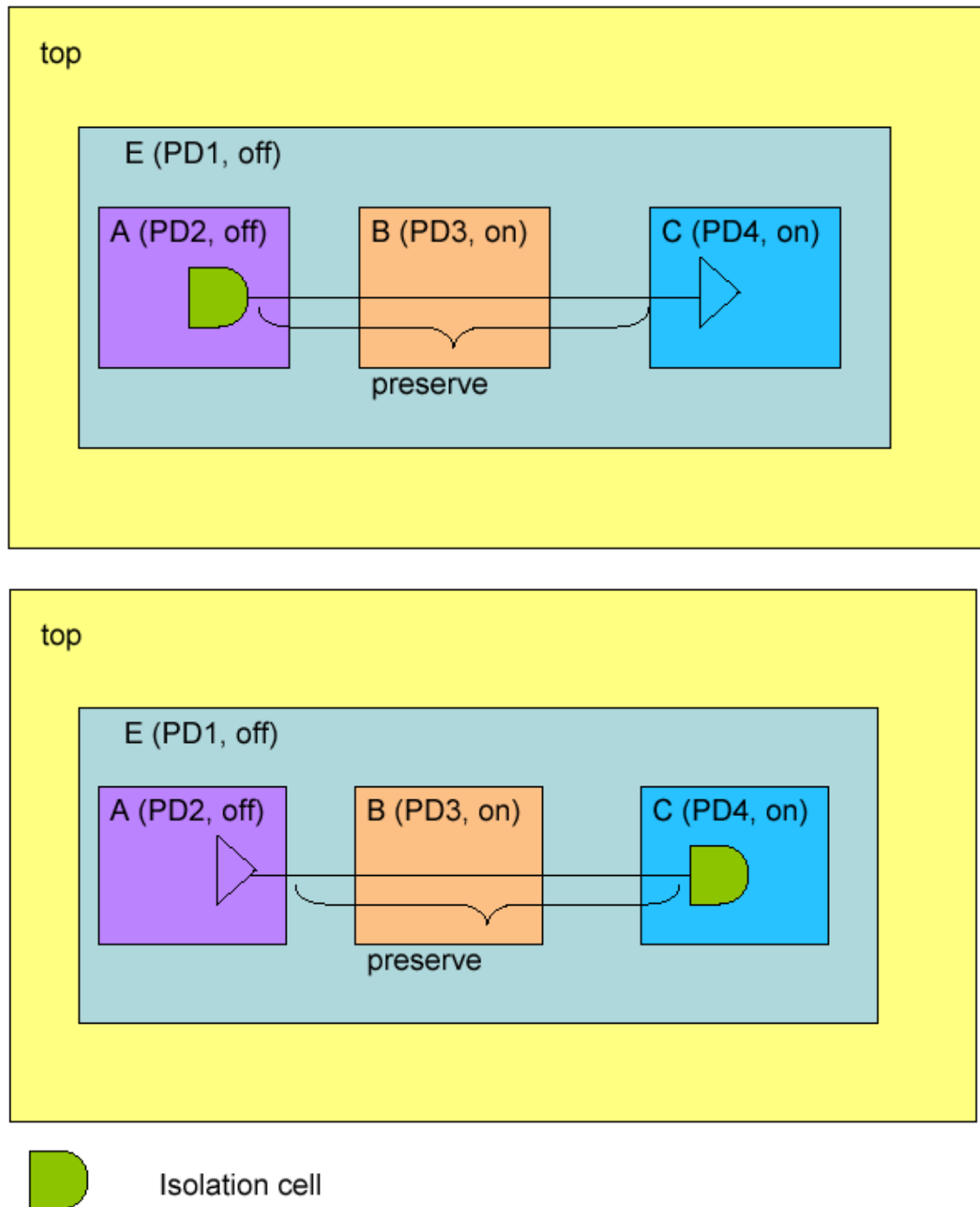
- on the driver (input) side of the isolation cell it crosses a power domain which is different from the originating (or from) power domain

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

- on the load (output) side of the isolation cell, it crosses a power domain which is different from the destination (or to) power domain

Figure 12-7 Preserving Nets to Prevent Buffer Insertion



Verify Added Power Logic

Verifies whether the level shifter that were inserted in the design conform to the level shifter rules in the loaded CPF file. The tool will flag if there are any missing level shifter cells or if the level shifter cells are not connected appropriately.

```
verify power structure -lvl
```

Note: To run this command you need to have access to version 7.1 (or later) of the Encounter[®] Conformal[®] Low Power software.

Run Incremental Optimization

Inserting the level shifters can have an impact on the timing.

- To fix any timing issues, run incremental optimization:

```
synthesize -incremental
```

Annotate switching activities

- To get accurate power analysis results, read in the switching activity information generated from the mapped netlist:

```
read_tcf tcf_file_for_mapped_netlist
```

Analyze Power

- After you have specified the switching activities, analyze the power using the following command:

```
report power
```

For more information, refer to [Chapter 5, “Power Analysis.”](#)

Analyze Design

After synthesizing the design, you can generate detailed timing and area reports using the various `report` commands. For more information, see [Analyze Design](#) in the [Recommended Flow](#).

For more information on generating reports for analysis, see [Generating Reports](#) in *Using Encounter RTL Compiler* and [“Analysis Commands”](#) in the *Command Reference for Encounter RTL Compiler*.

Most reports reflect information for the library domains (sets in CPF).

Export to Place and Route

- To export the necessary files for the Encounter place and route tool, use the following command:

```
write_encounter design [design] ....
```

This command writes the netlist file, the SDC constraint file, templates for the Encounter config file, Encounter setup file, and Encounter level-shifter table file. The directory and base name of the files are controlled through the `-basename` option of the `write_encounter` command. The default base name is `rc_enc_des`.

Important

This command creates *template* files. It is your responsibility to fill in the remaining information needed to start the Encounter place and route tool.

Encounter Tcl Script Template

The `write_encounter` command creates a setup template, `rc.enc_setup`. This file sets the following information (for MSV):

1. Loads in the Encounter configuration template that it created (see 1 in Figure 12-8).
2. Loads in the original CPF file (see 2 in Figure 12-8) and commits the CPF file.
3. Loads in an MSV-specific script —this file is empty for the CPF-flow.

Figure 12-8 Extract of a Sample Encounter Setup Template File

```
#####
#
# First Encounter setup file
# Created by RTL Compiler (RC) on date time
#
#####

...
# Design Import
#####
...
loadConfig rc_enc_des/rc.conf ← 1

# Mode Setup
#####
source rc_enc_des/rc.mode

# MSV Setup
#####
loadCPF /home/ria/rc_ex/cpf/ls_insert/my.cpf ← 2
commitCPF
source rc_enc_des/rc.msv.tcl
....
```

Encounter Config Template

The `write_encounter` command creates a template for the Encounter config file, `rc.conf`.

The following information is filled in the config file (as shown in Figure 12-9):

- The name of the netlist file (1)
- The type of the netlist file (2)
- The names of the timing libraries (3)
- The name of the constraint file (4)
- The list of the LEF files specified through the `-lef` option or read in during the RTL Compiler session (5)
- The power net names that RTL Compiler created based on the domain names (6). The power and net names are also used in the command file.

Figure 12-9 Sample Encounter Config Template File

```
#####
#
# First Encounter input configuration file
# Created by RTL Compiler (RC) on date time
#
#####
global rda_Input
set cwd lpwd
set rda_Input(ui_netlist) {rc_enc_des/rc.v} ← 1
set rda_Input(ui_netlisttype) {Verilog} ← 2
set rda_Input(ui_settop) {1}
set rda_Input(ui_topcell) {top}
set rda_Input(ui_timelib) {/my_path/lib1_1v08.lib \
...} ← 3
set rda_Input(ui_timingcon_file) {rc_enc_des/rc.sdc} ← 4
set rda_Input(ui_buf_footprint) {BUFX2MTH}
set rda_Input(ui_inv_footprint) {INVX12MTH}
set rda_Input(ui_leffile) {} ← 5
set rda_Input(ui_cts_cell_list) {CLKBUFX8MTH ...}
set rda_Input(ui_core_cntl) {aspect}
set rda_Input(ui_aspect_ratio) {1.0000}
set rda_Input(ui_captbl_file) {}
set rda_Input(ui_defcap_scale) {1.0}
set rda_Input(ui_res_scale) {1.0}
set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_pwrnet) {VDD_PD1 VDD_PD2 ....} ← 6
set rda_Input(ui_gen_footprint) {1}

##### RTL #####
#set rda_Input(ui_netlisttype) {RTL}
```

Low Power in Encounter RTL Compiler

Using CPF for Designs Using Dynamic Voltage Frequency Scaling

Other Flows

- [Recommended Bottom-Up Clock Gating Flow with DFT](#) on page 288
- [Scan Insertion after Clock-Gating Insertion](#) on page 292
- [Power Optimization when Starting with Mapped Netlist](#) on page 295
- [State-Retention Cell Replacement when Starting with Mapped Netlist](#) on page 297

Recommended Bottom-Up Clock Gating Flow with DFT

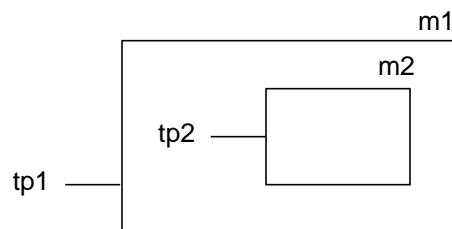
This flow is recommended when different teams are designing different portions of the design:

- The teams designing the blocks must insert the clock-gating logic in their block and ensure that the test pins of the clock-gating logic gets connected.
- The team assembling the design (working on the top level), is responsible for decloning the clock-gating logic (if desired), insertion of the observability logic and connection of the scan chains.

To illustrate this flow, consider a hierarchical design with top module `m1`. Assume module `m1` instantiates module `m2`. In the bottom-up flow, you first synthesize module `m2`, plug it into module `m1` and finally synthesize module `m1`. For this example, assume that

- The test port for module `m2` is defined as `tp2`
- The test port for module `m1` is defined as `tp1`

Figure 13-1 Example Design



In this flow, the test signals for the subblocks (module `m2` in the example) are automatically connected during Block-Level Synthesis.

The test pins of the clock-gating instances at the top-level (module `m1` in the example) are automatically connected to the top-level test port (`tp1` in the example) during Top-Level Synthesis (synthesis of module `m1` in the example).

The test ports on the subblocks need to be connected to the test port of the top level using the `edit_netlist connect` command (as illustrated in the Top-Level Synthesis section).

Note: For more information on DFT-specific steps, refer to *Design for Test in Encounter RTL Compiler*.

Block-Level Synthesis

In this example, you synthesize module `m2` and connect the test pins in `m2` to test port `tp2`.

1. General setup.

```
set_attribute lib_search_path ...
```

2. Read the target libraries.

```
set_attribute library library_list /
```

3. Enable clock-gating insertion.

```
set_attribute lp_insert_clock_gating true /
```

4. Read the HDL files of the module `m2`.

```
read_hdl m2.v ...
```

5. Elaborate the block.

```
elaborate
```

6. Set the timing and design constraints for module `m2`.

7. (Optional) Specify attributes to select clock-gating logic and to control clock gating.
Select a clock-gating cell that contains observability logic by setting the following design attribute:

```
set_attribute lp_clock_gating_add_obs_port true /designs/m2
```

8. Specify the DFT setup.

```
define_dft shift_enable -name se_signal -active high se_port  
define_dft test_mode -name TP2 -active high tp2
```

9. Identify the test signal object used for the clock-gating logic to the RC-LP engine.

```
set_att lp_clock_gating_test_signal /des*/m2/dft/test_signals/TP2 /designs/m2
```

10. Generate a generic netlist to remove any dont-care cells.

```
synthesize -to_generic
```

11. Run the DFT rule checker.

```
check_dft_rules  
report dft_registers
```

12. (Optional) Fix DFT rule violations.

```
fix_dft_violations...
```

13. Map to cells of the technology library and optimize the design.

```
synthesize -to_mapped
```

Note: At this time the RC-LP engine inserts the clock-gating logic and auto-connects the test pins. All flip-flops passing the DFT rule checker are mapped to scan flip-flops.

14. Write the structural netlist of module m2.

```
write_hdl> m2_mapped.v
```

Top-Level Synthesis

In the example, you synthesize module m1 and connect the test pins and test port tp2 to test port tp1.

1. General setup.

```
set_attribute lib_search_path ...  
...
```

2. Read the target libraries.

```
set_attribute library library_list /
```

3. Enable clock-gating insertion.

```
set_attr lp_insert_clock_gating true /
```

4. Read the top-level netlist and the structured netlist of the block-level logic (of module m2).

```
read_hdl m1.v m2_mapped.v
```

5. Elaborate the design.

6. Set the timing and design constraints for module m1.

7. (Optional) Specify attributes to select clock-gating cells and to control clock gating.

Select a clock-gating cell that contains observability logic by setting the following design attribute:

```
set_attribute lp_clock_gating_add_obs_port true /designs/m1
```

8. Specify the DFT setup and read in the scan abstract model for the logic abstract model that you read in.

```
define_dft shift_enable -name se_signal -active high se_port...  
define_dft test_mode -name TP1 -active high tp1 -create_port ...
```

9. Identify the test signal object used for the clock-gating logic to the RC-LP engine.

```
set_attr lp_clock_gating_test_signal /des*/m1/dft/test_signals/TP1 /des*/m1
```

10. Connect the top-level test port to the sub-block level test port.

```
edit_netlist connect tp1 [find / -pin i1/tp2]
```

This example assumes that the instance name of m2 in m1 is i1.

11. Generate a generic netlist to remove any dont-care cells.

```
synthesize -to_generic
```

Low Power in Encounter RTL Compiler

Other Flows

12. Run the DFT rule checker.

13. (Optional) Fix DFT rule violations.

14. Prevent boundary optimization on the `obs` port of the clock-gating logic in the block `m2`:

```
set_attr boundary_opto false [get_attr subdesign [find / -inst i1/RC_CG_HIER*]]
```

15. Map to cells of the technology library and optimize the design.

```
synthesize -to_mapped
```

Note: At this time the RC-LP engine inserts the clock-gating logic and auto-connects test pins of the clock-gating instances in module `m1` to test signal `tp1`. All flip-flops passing the DFT rule checker are mapped to scan flip-flops.

16. Declone the clock-gating logic.

```
clock_gating declone -hier
```

17. Insert the observability logic (X-OR tree and observation flops) for the clock-gating instances.

```
clock_gating insert_obs -hier...
```

18. Run the DFT rule checker to determine the DFT status of the observability logic.

```
check_dft_rules
```

19. Configure the scan chains.

20. Connect the scan chains.

```
connect_scan_chains -auto ...
```

21. To fix any timing issues introduced by inserting the observability logic and by connecting the scan chains, run incremental optimization:

```
synthesize -incremental
```

22. Write the structural netlist of module `m1`.

```
write_hdl > m1_mapped.v
```

23. Export the design.

Scan Insertion after Clock-Gating Insertion

This approach applies when different teams handle different aspects of the design.

In this case, one team already inserted clock gating and performed timing optimization on the entire design. This flow describes the remaining steps for the DFT team to perform scan insertion on the entire design.

For more information on DFT-specific steps, refer to *Design for Test in Encounter RTL Compiler*.

Important

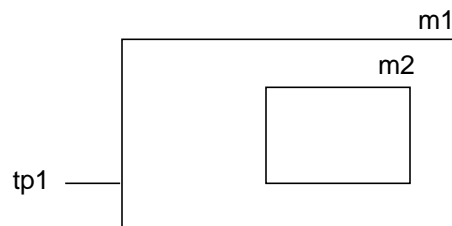
For this flow, the following assumptions are made:

- ☐ The first team inserted clock-gating logic which includes test logic and an observability port.
- ☐ The test pins of the clock-gating logic have not yet been connected.
- ☐ The logic to observe the clock-gating logic has not yet been inserted.

To illustrate this flow, consider the hierarchical design in Figure 13-2. Module `m1` instantiates module `m2`.

- `design_clock_gated.v` is the structural netlist containing the clock-gating logic
- `tp1` is the test port for module `m1`

Figure 13-2 Example Design



Low Power in Encounter RTL Compiler

Other Flows

To connect the test pins of the clock-gating logic, insert the observability logic for the clock-gating logic, configure and connect the scan chains for the entire design, follow these steps:

1. General set up.

2. Read the target libraries.

```
set_attribute library library_list /
```

3. Read the (non-scan) structural netlist of the design.

```
read_hdl -netlist design_clock_gated.v
```

4. Set the timing and design constraints.

5. Specify the DFT setup.

```
define_dft shift_enable -name se_signal -active high se_port ...
define_dft test_mode -name TP1 -active high tpl ...
```

6. Identify the test-control signal used for the clock-gating logic to the RC-LP engine.

```
set_att lp_clock_gating_test_signal /des*/m1/dft/test_signals/TP1 /designs/m1
```

Specify a test-control signal that was already defined with either the `define_dft test_mode` or the `define_dft shift_enable` command.

7. Connect the test pins of the clock-gating logic to the test signal.

```
clock_gating_connect_test
```

8. Insert the observability logic of the clock-gating instances in the design.

```
clock_gating_insert_obs...
```

9. Run the DFT rule checker.

```
check_dft_rules
report dft_registers
```

10. (Optional) Fix DFT rule violations.

```
fix_dft_violations...
```

11. Convert the flip-flops which pass the DFT rule checker to their scan flip-flops.

```
replace_scan
```

12. Configure and connect the top-level chains.

```
define_dft scan_chain ...
connect_scan_chains -auto_create_chains
```

13. To fix any timing issues introduced by inserting the observability logic and by connecting the scan chains, run incremental optimization:

```
synthesize -incremental
```

14. Write the structural netlist which includes the top-level scan chain connections.

Low Power in Encounter RTL Compiler

Other Flows

```
write_hdl > top_level_design_scan.v.
```

15. Export the design.

Power Optimization when Starting with Mapped Netlist

This approach applies when you performed synthesis with a third-party tool, and you want to use RTL Compiler to optimize timing and power in the netlist generated by this third-party tool. When starting with a structural netlist, perform the following steps:

1. General setup

```
set_attribute lib_search_path ...  
set_attribute hdl_search_path ...
```

2. Read the target libraries.

```
set_attribute library library_list /
```

3. Read the structural netlist.

```
read_hdl -netlist mapped_netlist
```

4. (Optional) Set timing and design constraints.

5. Specify leakage power constraint.

```
set_attribute max_leakage_power constraint /designs/design
```

6. (Optional) Specify leakage power optimization directive.

```
set_attribute lp_multi_vt_optimization_effort {low|medium|high} /des*/design
```

7. Specify dynamic power constraint.

```
set_attribute max_dynamic_power constraint /designs/design
```

8. (Optional) Control the power optimization using either:

```
☐ set_attribute lp_optimize_dynamic_power_first true /designs/design
```

```
☐ set_attribute lp_power_optimization_weight weight /designs/design
```

9. (Optional) Specify effort to use for propagation of switching activities

```
set_attribute lp_power_analysis_effort {low|medium|high} /
```

10. Run simulation on netlist.

11. Annotate switching activities.

```
read_tcf tcf_file_before_synthesis
```

12. Run incremental optimization

```
synthesize -incremental
```

13. Write out netlist for simulation

```
write -mapped netlist.v
```

Low Power in Encounter RTL Compiler

Other Flows

14. Run simulation on netlist.

15. Annotate switching activities.

```
read_tcf tcf_file_after_synthesis
```

16. Analyze power.

```
report power
```

Important

To get the most accurate power optimization and power analysis, you should simulate the design to generate a valid TCF. However, if you want to skip simulation or skip annotating activities, the RC-LP engine will use the default settings and propagate the switching activities from the primary inputs to the primary outputs on those nets that have no switching information asserted.

State-Retention Cell Replacement when Starting with Mapped Netlist

When starting with a structural netlist, perform the following steps to replace the sequential cells with state-retention cells.

1. General setup

```
set_attribute lib_search_path ...  
set_attribute hdl_search_path ...
```

2. Read the target libraries from the CPF file.

```
read_cpf -library cpf_file
```

Note: For each *library set* defined in the CPF file, RTL Compiler creates a subdirectory with that name in the `library_domains` directory. The libraries in each set are stored in the corresponding library domain.

3. Read the structural netlist.

```
read_hdl -netlist mapped_netlist
```

4. (Optional) Set timing and design constraints.

5. Read the power intent of the design.

```
read_cpf cpf_file
```

6. Check the quality of the CPF file.

```
check_cpf -ret
```

7. Replace the sequential cells with their equivalent state-retention cells.

```
state_retention_swap [-hierarchical] [-start_from instance] \  
[-connect_power_gating_pins]
```

8. Connect the power gating pins if they were not connected during the previous step.

```
state_retention_connect_power_gating_pins
```

9. Export the design.

Low Power in Encounter RTL Compiler

Other Flows

CPF Support in RTL Compiler

- [Migrating from the Native Flow to the CPF Flow](#) on page 300
- [Supported CPF Commands](#) on page 301
- [Mapping of CPF Commands to RTL Compiler Commands and Attributes](#) on page 302
 - [Library Cell-Related CPF Commands](#) on page 303
 - [Scope Commands](#) on page 304
 - [General Purpose Commands](#) on page 305
 - [Design Specification](#) on page 305

Migrating from the Native Flow to the CPF Flow

Native Command	Command in CPF file	RC command
create_library_domain	define_library_set	read_cpf
create_power_domain	create_power_domain	read_cpf
create_mode	create_power_mode, update_power_mode	read_cpf
create_power_ground_nets	create_power_nets, create_ground_nets	read_cpf
isolation_rule define	create_isolation_rule, update_isolation_rules	read_cpf
isolation_cell insert		commit_cpf -isolation_cell_only
isolation_cell import	identify_power_logic	read_cpf
level_shifter insert	create_level_shifter_rule	commit_cpf -level_shifter_only
level_shifter update		reload_cpf, commit_cpf -level_shifter_only
level_shifter check		verify_power_structure -lvl
state_retention define_driver	create_state_retention_rule, update_state_retention_rule	read_cpf
state_retention define_map	create_state_retention_rule, update_state_retention_rule	read_cpf

Supported CPF Commands

- Currently Not Supported Commands
- Commands Not Applicable to Synthesis
- Mapping of CPF Commands to RTL Compiler Commands and Attributes

Currently Not Supported Commands

```
create_global_connection  
create_ground_nets  
create_power_nets  
define_open_source_input_pin  
define_power_clamp_cell  
define_power_switch_cell  
create_power_switch_rule  
update_power_switch_rule
```

Commands Not Applicable to Synthesis

```
create_analysis_view  
create_bias_net  
create_mode_transition  
create_operating_corner
```

Mapping of CPF Commands to RTL Compiler Commands and Attributes

- [CPF Version Specification](#)
- [Library Specification](#)
- [Library Cell-Related CPF Commands](#)
- [Scope Commands](#)
- [General Purpose Commands](#)
- [Design Specification](#)

CPF Version Specification

CPF Command:	Encounter RTL Compiler Equivalent
<code>set_cpf_version [value]</code>	RTL Compiler issues an error if an invalid CPF version is specified. Note: Currently, only version 1.0 is supported.

Library Specification

CPF Command:	Encounter RTL Compiler Equivalent
<code>define_library_set</code> <code>-name library_set</code> <code>-libraries library_list</code>	<code>create_library_domain -name</code> <code>library_domain</code> where <i>library_domain</i> corresponds to <i>library_set</i> in the <code>define_library_set</code> command <code>set_attribute library library_list</code> <code>library_domain</code>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

Library Cell-Related CPF Commands

CPF Command:	Encounter RTL Compiler Equivalent
<pre> define_always_on_cell -cells <i>cell_list</i> [-library_set <i>library_set</i>] [[-power_switchable <i>LEF_power_pin</i> -ground_switchable <i>LEF_ground_pin</i>] -power <i>LEF_power_pin</i> -ground <i>LEF_ground_pin</i>] </pre>	<p>RTL Compiler annotates the library cells specified through -cells option with the following attribute:</p> <pre>set_attribute is_always_on true libcell</pre> <p>Unsupported options:</p> <p>-power_switchable, -ground_switchable, -power, -ground</p>
<pre> define_isolation_cell -cells <i>cell_list</i> [-library_set <i>library_set</i>] [-always_on_pin <i>pin_list</i>] [{ -power_switchable <i>LEF_power_pin</i> -ground_switchable <i>LEF_ground_pin</i> } -power <i>LEF_power_pin</i> -ground <i>LEF_ground_pin</i>] [-valid_location { from to}] [-non_dedicated] -enable <i>pin</i> </pre>	<p>RTL Compiler annotates the library cells specified through -cells option with the following attributes:</p> <pre>set_attribute is_isolation_cell true libcell</pre> <pre>set_attribute isolation_cell_enable_pin true libpin</pre> <p>Following options are handled internally by RTL Compiler:</p> <p>-always_on_pin</p> <p>Unsupported options:</p> <p>-power_switchable, -ground_switchable, -power, -ground, -valid_location, -non_dedicated</p>
<pre> define_level_shifter_cell -cells <i>cell_list</i> [-library_set <i>library_set</i>] [-always_on_pin <i>pin_list</i>] -input_voltage_range {voltage voltage_range} -output_voltage_range {voltage voltage_range} [-direction {up down bidir}] [-output_voltage_input_pin <i>pin</i>] { -input_power_pin <i>LEF_power_pin</i> [-output_power_pin <i>LEF_power_pin</i>] [-input_power_pin <i>LEF_power_pin</i>] -output_power_pin <i>LEF_power_pin</i> } -ground <i>LEF_ground_pin</i> [-valid_location { from to}] </pre>	<p>RTL Compiler annotates the library cells specified through -cells option with the following attributes:</p> <pre>set_attribute is_level_shifter true libcell</pre> <p>Following options are handled internally by RTL Compiler:</p> <p>-always_on_pin, -direction, -input_voltage_range, -output_voltage_range, -valid_location</p> <p>Unsupported options:</p> <p>-output_voltage_input_pin, -input_power_pin, -output_power_pin, -ground,</p>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

CPF Command:	Encounter RTL Compiler Equivalent
<pre>define_state_retention_cell -cells cell_list [-library_set library_set] [-always_on_pin pin_list] [-clock_pin pin] -restore_function expression [-restore_check expression] [-save_function expression] [-save_check expression] [{ -power_switchable LEF_power_pin -ground_switchable LEF_ground_pin } -power LEF_power_pin -ground LEF_ground_pin]</pre>	<p>RTL Compiler annotates the library cells specified through -cells option with the following attributes:</p> <pre>set_attribute power_gating_cell true libcell set_attribute power_gating_pin_class string libpin set_attribute power_gating_pin_phase {active_low active_high none} libpin</pre> <p>Following options are handled internally by RTL Compiler:</p> <pre>-always_on_pin</pre> <p>Unsupported options:</p> <pre>-restore_check, save_check, -power_switchable, -ground_switchable, -power, -ground</pre>

Scope Commands

CPF Command:	Encounter RTL Compiler Equivalent
<pre>set_design module [-ports port_list]</pre>	<p>For the first set_design in the main CPF: sets the scope to the top design</p> <p>For all other set_design commands, checks if the argument matches the subdesign of the current scope.</p> <p>RTL Compiler connects the ports specified through the -ports option of the set_design command to the drivers specified through the -port_mapping option of the set_instance command.</p>
<pre>set_instance [hier_instance [-merge_default_domains] [-port_mapping port_mapping_list]]</pre>	<p>Descends to the instance that corresponds to the instance referred to by set_instance command:</p> <pre>cd /designs/design/*/instances_hier/instance</pre> <p>RTL Compiler connects the ports specified by set_design -ports to the specified drivers.</p>
<pre>end_design</pre>	<p>changes the current scope to the previous scope: that is the top design or the parent instance where the preceding set_instance command was given.</p>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

General Purpose Commands

CPF Command:	Encounter RTL Compiler Equivalent
<code>set_array_naming_style [string]</code>	RTL Compiler checks if the style corresponds to the style set by the <code>hdl_array_naming_style</code> attribute. If the styles do not correspond, RTL Compiler issues an error because RTL Compiler cannot change the style after the design has been elaborated.
<code>set_hierarchy_separator [character]</code>	RTL Compiler uses the specified separator to parse the objects in the CPF file and convert them to appropriate RC objects
<code>set_power_unit [pW nW uW mW W]</code>	<code>set_attr lp_power_unit value /</code> Note: Not scope-sensitive in RTL Compiler, only the first command is taken into account in RTL Compiler.
<code>set_register_naming_style [string%s]</code>	RTL Compiler checks if the style corresponds to the style set by the <code>hdl_reg_naming_style</code> attribute. If the styles do not correspond, RTL Compiler issues an error because RTL Compiler cannot change the style after the design has been elaborated.
<code>set_time_unit [ns us ms]</code>	<code>set_attr lp_toggle_rate_unit value /</code> Note: Not scope-sensitive in RTL Compiler, only the first command is taken into account in RTL Compiler.

Design Specification

CPF Command:	Encounter RTL Compiler Equivalent
<pre>create_isolation_rule -name string -isolation_condition expression {-pins pin_list -from power_domain_list -to power_domain_list}... [-isolation_target {from to}] [-isolation_output {high low hold}] [-exclude pin_list]</pre>	<pre>isolation_rule define -name rule_name -enable_driver {port pin subport} -enable_polarity {active_low active_high} [-pins pin_list -from power_domain power_domain_list -to power_domain power_domain_list]... [-off_domain {from to}] [-output_value {high low hold}]</pre> <p>Following options are handled internally by RTL Compiler:</p> <pre>-exclude</pre>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

<pre>create_level_shifter_rule -name string {-pins pin_list -from power_domain_list -to power_domain_list}... [-exclude pin_list]</pre>	<pre>level_shifter insert [-from_library_domain domain_list] [-to_library_domain domain_list] [-instance_from instance...] [-instance_to instance...] [-location {from to}] [-dedicate_level_shifter libcell] [-prefix prefix] [-cpf_only]</pre> <p>Following options are handled internally by RTL Compiler:</p> <p>-exclude, -pins</p>
<pre>create_nominal_condition -name string -voltage float [-pmos_bias_voltage float] [-nmos_bias_voltage float]</pre>	<p>Following options are handled internally by RTL Compiler:</p> <p>-name, -voltage</p> <p>Unsupported options:</p> <p>-pmos_bias_voltage, -nmos_bias_voltage</p>
<pre>create_power_domain -name power_domain {-default [-instances instance_list] [-boundary_ports pin_list] -instances instance_list [-boundary_ports pin_list] -boundary_ports pin_list } [-shutoff_condition expression] [-default_restore_edge expression] [-default_save_edge expression] [-power_up_states {high low random}]</pre>	<pre>create_power_domain -name power_domain set_attribute default true power_domain set_attribute power_domain power_domain {instance pin port} set_attribute shutoff_signal {pin port bus} power_domain set_attribute shutoff_signal_polarity {shutoff_signal_polarity} power_domain</pre> <p>Note: RTL Compiler uses the -default_restore_edge and -default_save_edge options for the state_retention define_driver command, if the -restore_edge and -save_edge options would be missing from the create_state_retention_rule command.</p> <p>Unsupported options:</p> <p>-power_up_states</p>
<pre>create_power_mode -name string -domain_conditions domain_condition_list [-default]</pre>	<p>For each nominal condition specified with the default mode, RTL Compiler looks at the corresponding update_nominal_condition command to find the associated library set for that condition, and links the corresponding library domain to the instances that belong to the power domain associated with this condition.</p> <p>For a DVFS design (containing other modes that have different operating voltages compared to the voltages in default mode), RTL Compiler will set the following attribute for the other modes:</p> <pre>set_attribute library_domain_by_mode {{lib_domain mode}...} power_domain</pre>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

<pre>create_state_retention_rule -name string { -domain power_domain -instances instance_list } [-restore_edge expression [-save_edge expression]]</pre>	<pre>state_retention define_map [-power_domain power_domain] [-instances instance_list]</pre> <p>RTL Compiler only maps to cells specified through the <code>define_state_retention_cell</code> command for the state retention rules.</p> <pre>state_retention define_driver \ -driver {pin port} [-create_port] \ -driver_polarity {active_low active_high}</pre>
<pre>identify_always_on_driver -pins pin_list [-no_propagation]</pre>	<pre>set_attribute ideal_driver true {pin port}</pre> <p>If the <code>-no_propagation</code> option is specified, RTL Compiler marks the net connected to the pin or port as preserved.</p>
<pre>identify_power_logic -type isolation -instances instance_list</pre>	<pre>isolation_cell import</pre> <p>Following options are handled internally by RTL Compiler:</p> <pre>-instances</pre>
<pre>set_power_target { -leakage float -dynamic float -leakage float -dynamic float}</pre>	<pre>set_attribute max_leakage_power float design set_attribute max_dynamic_power float design</pre> <p>The attributes are set on the design.</p>
<pre>set_switching_activity { { -all -pins pin_list -instances instance_list [-hierarchical]} -probability float -toggle_rate float } [-clock_pins pin_list] -toggle_percentage float } [-mode mode]</pre>	<pre>set_attribute lp_asserted_probability float xxx set_attribute lp_asserted_toggle_rate float xxx</pre> <p>where xxx depends on the options specified with <code>set_switching_activity</code>:</p> <ul style="list-style-type: none"> ■ <code>-all</code>: xxx applies to all nets and pins ■ <code>-pins</code>: xxx applies to all specified pins ■ <code>-instances</code>: xxx applies to the output pins of these instances <p>Unsupported options:</p> <pre>-clock_pins, -toggle_percentage, -mode</pre>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

<pre>update_isolation_rules -names rule_list { -location {from to} -prefix string -combine_level_shifting -cells cell_list -library_set library_set -open_source_pins_only}...</pre>	<pre>isolation_rule define -name rule [-location {from to}] [-prefix string] [-cell cell_list]</pre> <p>If the <code>-combine_level_shifting</code> option is set, RTL Compiler will insert a library cell with the functionality of a level shifter and isolation cell if this type of cell is available in the library and if it is required.</p> <p>Unsupported options: <code>-open_source_pins_only</code></p>
<pre>update_level_shifter_rules -names rule_list { -location {from to} -cells cell_list -library_set library_set -prefix string }...</pre>	<pre>level_shifter insert [-from_library_domain library_domain] [-to_library_domain library_domain] [-location {from to}] [-dedicate_level_shifter libcell] [-prefix string]</pre> <p>where the from and to library domains are derived from the <code>create_level_shifter_rule</code> command.</p>
<pre>update_nominal_condition -name condition -library_set library_set</pre>	<p>Following options are handled internally by RTL Compiler: <code>-name</code>, <code>-library_set</code></p>
<pre>update_power_domain -name domain { -internal_power_net net -internal_ground_net net -min_power_up_time float -max_power_up_time float -pmos_bias_net net -nmos_bias_net net -user_attributes string_list -rail_mapping rail_mapping_list -library_set library_set} ...</pre>	<pre>create_power_ground_nets -name net -power -internal create_power_ground_nets -name net -ground -internal</pre> <p>Unsupported options: <code>-min_power_up_time</code>, <code>-max_power_up_time</code>, <code>-pmos_bias_net</code>, <code>-nmos_bias_net</code>, <code>-user_attributes</code>, <code>-rail_mapping</code>, <code>-library_set</code></p>
<pre>update_power_mode -name mode { -activity_file file -activity_file_weight weight -sdc_files sdc_file_list -peak_ir_drop_limit domain_voltage_list -average_ir_drop_limit domain_voltage_list -leakage_power_limit float -dynamic_power_limit float}...</pre>	<pre>read_tcf file read_sdc [-mode mode] files</pre> <p>Following options are handled internally by RTL Compiler: <code>-activity_file_weight</code></p> <p>Unsupported options: <code>-peak_ir_drop_limit</code>, <code>-average_ir_drop_limit</code>, <code>-leakage_power_limit</code>, <code>-dynamic_power_limit</code></p>

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

update_state_retention_rules

```
-names rule_list  
{-cell_type string | -cell libcell}  
-library_set library_set
```

RTL Compiler only maps to cells specified through the `define_state_retention_cell` command.

```
set_attribute lp_map_to_srpg_cells cell  
instance_list
```

where *instance_list* is derived from the corresponding `create_state_retention_rule` command

Following options are handled internally by RTL Compiler:

```
-library_set
```

Low Power in Encounter RTL Compiler

CPF Support in RTL Compiler

Low Power Synthesis Reference

- Commands
- Attributes

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Commands

Command	Related Commands	Related Attributes
clock_gating connect_test	report clock_gating	lp_clock_gating_test_signal
clock_gating declone	report clock_gating	
clock_gating import	report clock_gating	
clock_gating insert_in_netlist	report clock_gating	
clock_gating insert_obs	define clock report clock_gating	lp_clock_gating_add_obs_port
clock_gating join	report clock_gating	
clock_gating remove	report clock_gating	
clock_gating share	report clock_gating	
clock_gating split	report clock_gating	
read_saif	report power	lp_asserted_probability lp_probability_type lp_asserted_toggle_rate lp_toggle_rate_type
read_tcf	report power	lp_asserted_probability lp_asserted_toggle_rate lp_probability_type lp_toggle_rate_type
read_vcd	report power	lp_dynamic_analysis_scope
report clock_gating	synthesize clock_gating declone clock_gating insert_in_netlist clock_gating insert_obs clock_gating join clock_gating share clock_gating split clock_gating remove	lp_clock_gating_add_obs_port lp_clock_gating_add_reset lp_clock_gating_cell lp_clock_gating_control_point lp_clock_gating_exclude lp_clock_gating_max_flops lp_clock_gating_min_flops lp_clock_gating_style

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Command	Related Commands	Related Attributes
report gates -power	synthesize	cell_leakage_power leakage_power_scale_in_nW lp_asserted_probability lp_asserted_toggle_rate lp_computed_probability lp_computed_toggle_rate lp_default_probability lp_default_toggle_rate lp_leakage_power lp_net_power lp_power_unit
report operand_isolation		lp_insert_operand_isolation
report power	elaborate read_saif read_tcf synthesize	cell_leakage_power leakage_power_scale_in_nW lp_asserted_probability lp_asserted_toggle_rate lp_computed_probability lp_computed_toggle_rate lp_default_probability lp_default_toggle_percentage lp_default_toggle_rate lp_leakage_power lp_map_to_srpg_cells lp_net_power lp_power_unit lp_srpg_pg_driver
state_retention connect_power_gating_pins		
state_retention define_driver		lp_srpg_driver power_gating pin_class power_gating pin_phase
state_retention define_map		lp_map_to_srpg_type hdl_enable_proc_name hdl_proc_name

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Command	Related Commands	Related Attributes
state_retention swap	state_retention define_driver state_retention define_map	lp_map_to_srpg_type hdl_enable_proc_name hdl_proc_name
synthesize	report power	lp_map_to_srpg_cells lp_srpg_pg_driver max_dynamic_power max_leakage_power
write_forward_saif	read_saif	
write_saif	read_saif	
write_tcf	read_tcf	lp_asserted_probability lp_asserted_toggle_rate

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attributes

Attribute	Related Commands	Related Attributes
Root		
lp_clock_gating_prefix	synthesize report clock_gating	
lp_clock_gating_exceptions_aware		
lp_insert_clock_gating	synthesize report clock_gating	
lp_insert_operand_isolation	report operand_isolation	
lp_multi_vt_optimization_effort	synthesize report power	max_leakage_power
lp_operand_isolation_prefix	synthesize report operand_isolation	
lp_power_analysis_effort	synthesize report power	lp_asserted_probability lp_asserted_toggle_rate lp_computed_probability lp_computed_toggle_rate
lp_power_unit	report power	
lp_pso_aware_estimation		
lp_toggle_rate_unit	report power	lp_default_toggle_rate lp_toggle_rate
Design		
lp_clock_gating_add_obs_port	report clock_gating synthesize	lp_clock_gating_cell
lp_clock_gating_add_reset	synthesize	lp_clock_gating_cell
lp_clock_gating_cell	report clock_gating synthesize	
lp_clock_gating_control_point	report clock_gating synthesize	lp_clock_gating_cell
lp_clock_gating_exclude	report clock_gating synthesize	

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attribute	Related Commands	Related Attributes
lp_clock_gating_max_flops	synthesize	
lp_clock_gating_min_flops	synthesize	
lp_clock_gating_module	synthesize	
lp_clock_gating_style	report clock_gating synthesize	lp_clock_gating_cell
lp_clock_gating_test_signal	synthesize clock_gating connect_test clock_gating insert_obs	
lp_default_probability	report_power	lp_probability
lp_default_toggle_rate	report_power	lp_toggle_rate lp_toggle_rate_unit
lp_internal_power	report_power	lp_power_unit
lp_leakage_power	report power	lp_power_unit
lp_map_to_srpg_cells	synthesize report power	power_gating_cell power_gating_pin power_gating_pin_phase lp_srpg_pg_driver
lp_net_power	report_power	lp_power_unit
lp_optimize_dynamic_power_first	report gates -power report power synthesize	lp_power_optimization_weight
lp_power_optimization_weight	report gates -power report power synthesize	lp_optimize_dynamic_power_first
lp_srpg_pg_driver	synthesize report power	power_gating_cell power_gating_pin power_gating_pin_phase lp_map_to_srpg_cells
max_dynamic_power	report power	lp_power_unit
max_leakage_power	report power	lp_power_unit

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attribute	Related Commands	Related Attributes
SubDesign		
lp_clock_gating_cell	report clock_gating synthesize	
lp_clock_gating_exclude	report clock_gating synthesize	
lp_clock_gating_module	report clock_gating synthesize	
Library		
leakage_power_scale_in_nW	report power	
Libcell		
cell_leakage_power	report power	
power_gating_cell	report power	power_gating_cell_type power_gating_pin_class power_gating_pin_phase
power_gating_cell_type	report power	power_gating_cell power_gating_pin_class power_gating_pin_phase
Libpin		
power_gating_pin_class	report power	power_gating_cell power_gating_cell_type power_gating_pin_phase
power_gating_pin_phase	report power	power_gating_cell power_gating_cell_type power_gating_pin_class
Instance		
lp_clock_gating_cell	report clock_gating synthesize	
lp_clock_gating_exclude	report clock_gating synthesize	

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attribute	Related Commands	Related Attributes
lp_clock_gating_module	report clock_gating synthesize	
lp_default_probability	report power	lp_probability
lp_default_toggle_rate	report power	lp_toggle_rate lp_toggle_rate_unit
lp_internal_power	report power	lp_power_unit
lp_leakage_power	report power	lp_power_unit
lp_map_to_srpg_cells	synthesize report power	power_gating_cell power_gating_pin power_gating_pin_phase lp_srpg_pg_driver
lp_map_to_srpg_type		
lp_net_power	report power	lp_power_unit
lp_srpg_driver		
Net		
lp_asserted_probability	read_saif read_tcf read_vcd	lp_default_probability
lp_asserted_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_default_toggle_rate lp_toggle_rate_unit
lp_computed_probability	read_saif read_tcf read_vcd	lp_asserted_probability lp_default_probability
lp_computed_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_asserted_toggle_rate lp_default_toggle_rate lp_toggle_rate_unit
lp_net_power	report power	lp_power_unit

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attribute	Related Commands	Related Attributes
lp_probability_type	define_clock read_saif read_tcf read_vcd	
lp_toggle_rate_type	read_saif read_tcf read_vcd	
Pin		
lp_asserted_probability	read_saif read_tcf read_vcd	lp_default_probability
lp_asserted_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_default_toggle_rate lp_toggle_rate_unit
lp_computed_probability	read_saif read_tcf read_vcd	lp_asserted_probability lp_default_probability
lp_computed_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_asserted_toggle_rate lp_default_toggle_rate lp_toggle_rate_unit
lp_net_power	report_power	lp_power_unit
lp_probability_type	define_clock read_saif read_tcf read_vcd	
lp_toggle_rate_type	read_saif read_tcf read_vcd	

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attribute	Related Commands	Related Attributes
Port		
lp_asserted_probability	read_saif read_tcf read_vcd	lp_default_probability
lp_asserted_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_default_toggle_rate lp_toggle_rate_unit
lp_computed_probability	read_saif read_tcf read_vcd	lp_asserted_probability lp_default_probability
lp_computed_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_asserted_toggle_rate lp_default_toggle_rate lp_toggle_rate_unit
lp_net_power		lp_power_unit
lp_probability_type	define_clock read_saif read_tcf read_vcd	
lp_toggle_rate_type	read_saif read_tcf read_vcd	
Subport		
lp_asserted_probability	read_saif read_tcf read_vcd	lp_default_probability
lp_asserted_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_default_toggle_rate lp_toggle_rate_unit

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Attribute	Related Commands	Related Attributes
lp_computed_probability	read_saif read_tcf read_vcd	lp_asserted_probability lp_default_probability
lp_computed_toggle_rate	define_clock read_saif read_tcf read_vcd	lp_asserted_toggle_rate lp_default_toggle_rate lp_toggle_rate_unit
lp_net_power		lp_power_unit
lp_probability_type	define_clock read_saif read_tcf read_vcd	
lp_toggle_rate_type	read_saif read_tcf read_vcd	
Clock		
lp_default_toggle_percentage	report power	lp_toggle_rate_unit

Low Power in Encounter RTL Compiler

Low Power Synthesis Reference

Advanced Low Power Reference

- [Commands](#)
- [Attributes](#)

Low Power in Encounter RTL Compiler

Advanced Low Power Reference

Commands

Command	Related Commands	Related Attributes
create_library_domain		library_domain
commit_cpf	read_cpf	
isolation_cell remove		
level_shifter remove		
read_cpf		
reload_cpf	read_cpf	
report isolation	read_cpf	
report level_shifter	read_cpf	
report power_domain	read_cpf create_power_domain	power_domain
verify_power_structure	read_cpf	
write_encounter		

Low Power in Encounter RTL Compiler

Advanced Low Power Reference

Attributes

Attribute	Related Commands	Related Attributes
design		
base_mode		library_domain_by_mode
library_domain	create_library_domain synthesize	
hdl_arch		
library_domain		
instance		
inherited_default_power_domain		
library_domain		
power_domain		
isolation_rule		
cells	read_cpf	
cpf_pins	read_cpf	
enable_driver	read_cpf	
enable_polarity	read_cpf	
exclude_pins	read_cpf	
from_power_domain	read_cpf	
location	read_cpf	
off_domain	read_cpf	
output_value	read_cpf	
pins	commit_cpf	
prefix	read_cpf	
to_power_domain	read_cpf	
level_shifter_group		
direction	read_cpf	library
level_shifter_cells	read_cpf	library
max_input_voltage	read_cpf	library

Low Power in Encounter RTL Compiler

Advanced Low Power Reference

Attribute	Related Commands	Related Attributes
max_output_voltage	read_cpf	library
min_input_voltage	read_cpf	library
min_output_voltage	read_cpf	library
valid_location	read_cpf	library
level_shifter_rule		
cells	read_cpf	
cpf_pins	read_cpf	
exclude_pins	read_cpf	
from_power_domain	read_cpf	
location	read_cpf	
pins	commit_cpf	
prefix	read_cpf	
to_power_domain	read_cpf	
library_domain		
active_operating_conditions		
is_default		
library		
power_library	report power	
operating_conditions		
wireload_selection		
libcell		
is_isolation_cell		
is_level_shifter		
libpin		
isolation_cell_enable_pin		
level_shifter_enable_pin		
pin		
inherited_default_power_domain		
isolation_rule	commit_cpf	

Low Power in Encounter RTL Compiler

Advanced Low Power Reference

Attribute	Related Commands	Related Attributes
power_domain		
port		
inherited_default_power_domain		
isolation_rule	commit_cpf	
power_domain		
power_domain		
default	create_power_domain	
dft_iso_rule		
library_domain		
library_domain_by_mode		
shutoff_signal		
shutoff_signal_polarity		
power_ground_net		
internal	read_cpf	
rail_connection	read_cpf	
subdesign		
library_domain		

Low Power in Encounter RTL Compiler

Advanced Low Power Reference

Index

A

asynchronous reset logic, in clock-gating cell [122](#)

attributes

- clock-gating insertion
 - clock_gating_module [118](#)
 - lp_clock_gating_add_obs_port [123](#), [139](#), [289](#), [290](#)
 - lp_clock_gating_add_reset [122](#)
 - lp_clock_gating_cell [117](#)
 - lp_clock_gating_control_point [123](#), [135](#)
 - lp_clock_gating_exclude [130](#), [145](#)
 - lp_clock_gating_extract_common_enable [131](#), [222](#), [247](#), [276](#)
 - lp_clock_gating_max_flops [130](#)
 - lp_clock_gating_min_flops [130](#)
 - lp_clock_gating_module [117](#)
 - lp_clock_gating_prefix [114](#)
 - lp_clock_gating_style [122](#)
 - lp_clock_gating_test_signal [136](#), [137](#)
 - lp_insert_clock_gating [114](#)
- dynamic power optimization
 - lp_optimize_dynamic_power_first [1](#) [94](#)
 - max_dynamic_power [204](#), [216](#), [241](#)
- leakage power optimization
 - leakage_power_scale_in_nW [51](#)
 - lp_multi_vt_optimization_effort [193](#)
 - max_leakage_power [192](#), [216](#), [241](#)
- operand-isolation insertion
 - lp_insert_operand_isolation [180](#)
 - lp_operand_isolation_prefix [180](#)
- power analysis
 - lp_asserted_probability [58](#)
 - lp_asserted_toggle_rate [58](#)
 - lp_computed_probability [58](#), [101](#)
 - lp_computed_toggle_rate [58](#), [101](#)
 - lp_default_probability [61](#), [99](#)
 - lp_default_toggle_rate [61](#), [99](#)
 - lp_internal_power [99](#), [208](#)
 - lp_leakage_power [99](#), [198](#)
 - lp_net_power [99](#)
 - lp_power_analysis_effort [61](#)

lp_power_unit [192](#), [204](#)

lp_probability_type [63](#)

lp_toggle_rate_type [63](#)

C

clock gating

- concept [106](#)
- enabling [114](#)
- inserting in mapped netlist [170](#)
- limitations [174](#)
- preventing [130](#)
- removing [156](#)
- reporting [146](#)
- retiming, using with [108](#)
- status [144](#)
- troubleshooting [144](#), [173](#)

clock-gating logic (CG logic)

- customized logic, using [117](#)
- fanout, controlling [130](#)
- flip-flops with synchronous reset, handling of [111](#), [112](#), [118](#)

instances

- location in design hierarchy [116](#), [224](#), [251](#), [279](#)
- merging [152](#)
- splitting in multiple levels [164](#)

integrated clock-gating cell, using [122](#)

latch or non-latch gating logic [123](#)

location of test-control logic in CG cell [135](#)

observability logic, selecting with [139](#), [289](#), [290](#)

port names, list of [116](#)

prefix for modules, nets, ports [114](#)

sharing enable [159](#)

subdesign

- location in design hierarchy [115](#), [224](#), [251](#), [279](#)

commands

- clock_gating connect_test [138](#)
- clock_gating declone [152](#)
- clock_gating insert_in_netlist [171](#)
- clock_gating insert_obs [139](#)
- clock_gating join [169](#)

- clock_gating remove [156](#)
- clock_gating share [159](#)
- clock_gating split [164](#)
- define_dft test_mode [137](#)
- read_saif [71](#)
- read_tcf [64](#)
- read_vcd [72](#)
- report clock_gating [146](#)
- report gates [98](#)
- report instance [97](#)
- report level_shifter [224](#), [251](#), [278](#)
- report power [92](#)
- state_retention swap [297](#)
- ungroup [116](#), [180](#)
- write_encounter [230](#), [256](#), [284](#)
- write_tcf [102](#)
- compression logic
 - previewing changes [108](#)

D

- design information hierarchy [215](#), [239](#)
- DFT, clock-gating flow with
 - bottom-up flow [288](#)
 - starting from mapped netlist [170](#)
 - top-down flow [141](#)
- DVFS designs [33](#), [260](#)
- dynamic power
 - optimization
 - enabling [204](#)
 - flow [201](#)
 - preventing [205](#)
 - reporting [207](#)

E

- enable signal
 - is constant, message [144](#)
 - removing glitch [123](#)

F

- files, required [34](#)
- flows
 - clock gating with retiming flow [108](#)
 - clock-gating flow supporting DFT
 - bottom-up flow [288](#)
 - top-down flow [141](#)

- dynamic power optimization [201](#)
- leakage power optimization [189](#)
- scan insertion after clock-gating
 - insertion [292](#)
- state-retention cell swapping after
 - mapping [297](#)
- top-down low power synthesis flow [38](#)

I

- instance power, reporting [100](#)
- integrated clock-gating cell
 - default type [123](#)
 - glitch on enable signal, handling [123](#)
 - library cell name, specifying [117](#)
 - type, selecting [122](#)
- internal power
 - calculation for cell [52](#)
 - concept [49](#)
 - reporting for instance [100](#)

L

- leakage power
 - calculation for cell [54](#)
 - concept [48](#)
 - optimization
 - enabling [192](#)
 - flow [189](#)
 - preventing [195](#)
 - reporting [198](#)
 - reporting for instance [100](#)
 - units [51](#), [198](#)
- level shifters
 - reporting [224](#), [278](#)
- library requirements
 - clock-gating insertion [98](#), [122](#)
 - dynamic power optimization [203](#)
 - leakage power optimization [191](#)
- log file, messages
 - during clock-gating insertion [144](#)
 - during dynamic power optimization [206](#)
 - during leakage power optimization [196](#)
 - during operand isolation [181](#)
 - reading switching activities [80](#)

M

MSV information
 in design hierarchy [215](#), [239](#)
multi-stage clock-gating logic
 consolidating [169](#)
 creating [159](#), [164](#)

N

net power
 calculation [54](#)
 components [50](#)
 reporting [100](#)

O

operand isolation (OI)
 enabling [180](#)
 limitations [185](#)
 reporting [182](#)
 status [181](#)
operand-isolation logic (OI logic)
 location in design hierarchy [180](#)
 prefix for modules, instances, ports [180](#)

P

power analysis
 by cell type [98](#)
 flow [85](#)
 of design [86](#), [89](#), [92](#)
 of leaf instances [93](#)
 of nets [96](#)
 RTL specific [86](#)
 sorting report [94](#)
 troubleshooting [102](#)
power dissipation, of design
 calculation [51](#)
 modelling [50](#)
 reporting [92](#)
power optimization
 controlling [194](#)
probability
 asserting [60](#)
 definition [58](#)
 retrieving [58](#), [101](#)

source of information [63](#)

R

report
 clock gating [146](#)
 level shifters [224](#), [278](#)
 operand isolation [182](#)
 power by cell type [98](#)
 power of hierarchical instances [92](#)
 power of leaf instances [97](#)
RTL power analysis [86](#)

S

SAIF file, reading [71](#)
short-circuit power, concept [49](#)
status
 clock gating insertion [144](#)
 operand isolation insertion [181](#)
switching activities
 propagation, controlling [62](#), [188](#), [200](#)
 reading from SAIF file [71](#)
 reading from TCF file [64](#)
 reading from VCD file [72](#)

T

TCF file
 changing clock frequency [65](#)
 reading [64](#)
 reading multiple files [66](#)
timing problems, fixing after scan
 connection [143](#), [226](#), [254](#), [282](#), [291](#), [293](#)
toggle rate
 asserting [60](#)
 retrieving [58](#), [101](#)
 source of information [63](#)
transition density. See toggle rate
troubleshooting
 clock gating [144](#), [173](#)
 messages when reading TCF/SAIF file [81](#)
 power analysis [102](#)

V

VCD file, reading [72](#)