
Cadence[®] Synthesis Rapid Adoption Kit

Product Version 5.0.10
May 2003

©2002-2003 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) 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 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. 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. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. 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>Preface</u>	7
<u>About This Manual</u>	7
<u>Other Information Sources</u>	7
<u>Documentation Conventions</u>	8
<u>Text Command Syntax</u>	8
<u>Using Menus</u>	9
<u>Using Forms</u>	9

1

<u>Before You Begin</u>	11
<u>Invoking BuildGates Synthesis or BuildGates Extreme Synthesis</u>	11
<u>Command Language Interface</u>	12
<u>Getting Started</u>	12
<u>Set up your UNIX Environment</u>	12
<u>Download the Tutorial Data</u>	12
<u>Install the Libraries</u>	13
<u>Regarding the Results</u>	13

2

<u>Getting Started with BuildGates Synthesis</u>	15
<u>Introduction to BuildGates Synthesis</u>	16
<u>Objectives</u>	16
<u>Example Design</u>	16
<u>Setting Up to Run the Tutorial</u>	18
<u>Required License</u>	18
<u>Required Files</u>	18
<u>Basic BuildGates Flow</u>	19
<u>Setting Up Your Design Environment</u>	20
<u>Setting Global Variables</u>	20
<u>Reading the Technology Library</u>	21

Cadence Synthesis Rapid Adoption Kit

<u>Reading the HDL Models for the Design</u>	23
<u>Creating a Generic Gate-Level Netlist</u>	24
<u>Setting Test Synthesis Assertions</u>	26
<u>Setting Constraints</u>	30
<u>Checking Constraints</u>	39
<u>Setting Path Groups</u>	42
<u>Optimizing the Design</u>	43
<u>Generating Reports</u>	45
<u>Writing Netlist and Database Files</u>	51
<u>Setting Scan Chain Assertions</u>	52
<u>Incremental Timing Optimization</u>	55
<u>Interfacing with Physical Design Tools</u>	59
<u>Using BuildGates Synthesis Graphical User Interface</u>	61
<u>Migrating to BuildGates from Design Compiler</u>	66
<u>Constraint Translation Example</u>	67
<u>Distributed Synthesis</u>	71
<u>Summary</u>	72
<u>Where to Go from Here</u>	72

3

<u>Getting Started with Design For Test</u>	73
<u>Introduction to BuildGates Test Synthesis</u>	74
<u>Objectives</u>	75
<u>Example Design</u>	75
<u>Setting Up to Run the Tutorial</u>	77
<u>Required License</u>	77
<u>Required Files</u>	77
<u>BuildGates Top-Down Test Synthesis Configuration Flow</u>	78
<u>Setting up the Design Environment</u>	80
<u>Read Libraries and the HDL Models for the Design</u>	81
<u>Setting Test Synthesis Assertions</u>	82
<u>Checking DFT Assertions and Rule Violations</u>	84
<u>Fixing DFT Rule Violations</u>	87
<u>Setting Constraints</u>	89
<u>Optimizing the Design</u>	89

Cadence Synthesis Rapid Adoption Kit

<u>Setting Scan Chain Assertions</u>	90
<u>Connecting the Scan Chains</u>	92
<u>Checking the Timing of the Design</u>	94
<u>Viewing the Scan Chains</u>	95
<u>Writing Netlist and Database Files</u>	96
<u>Writing Interface Files to Third Party ATPG Tools</u>	97
<u>Summary</u>	98

4

Getting Started with Datapath

<u>Introduction to BGX Datapath</u>	100
<u>Objectives</u>	101
<u>Example Design</u>	101
<u>Setting Up to Run the Tutorial</u>	102
<u>Required License</u>	102
<u>Required Files</u>	102
<u>Traditional Synthesis Run</u>	103
<u>BGX Datapath Flow</u>	104
<u>Setting up the Design Environment</u>	105
<u>Setting up Global Variables</u>	105
<u>Reading the Synthesis Technology Libraries</u>	106
<u>Reading the HDL Models for the Design</u>	106
<u>Creating a Generic Gate-Level Netlist</u>	106
<u>Setting Constraints</u>	107
<u>Operator Merging</u>	108
<u>Optimizing the Design</u>	111
<u>Generating Reports</u>	111
<u>Writing Netlist, Database Files and Report File</u>	114
<u>Examining Traditional Synthesis Results</u>	115
<u>Comparing BGX and Traditional Results</u>	115
<u>Summary</u>	116

5

Getting Started with Low Power Synthesis

<u>Introduction</u>	118
---------------------------	-----

Cadence Synthesis Rapid Adoption Kit

<u>Objectives</u>	118
<u>Example Design</u>	119
<u>Setting Up for This Module</u>	119
<u>Required License</u>	119
<u>Ensure Access to a Verilog Simulator</u>	120
<u>Link to the Simulation Library</u>	120
<u>Required Files</u>	120
<u>Regular BGX Flow with Power Estimation</u>	122
<u>BGX LPS Flow</u>	123
<u>Setting up Design Environment</u>	124
<u>Reading the Synthesis Technology Libraries</u>	124
<u>Reading the HDL Models for the Design</u>	125
<u>Creating Generic Gate-level Netlist and Exploring Sleep-mode Options at the RTL Level</u>	125
<u>Setting Constraints</u>	130
<u>Setting Sleep-Mode and Clock-Gating Options</u>	130
<u>Optimizing the Design</u>	132
<u>Simulating and Reading the Toggle Count File</u>	135
<u>Optimizing the Power</u>	137
<u>Analyzing the Results and Generating Reports</u>	137
<u>Power Analysis in BuildGates Extreme</u>	141
<u>Generating Reports</u>	148
<u>Writing Netlist and Database Files</u>	148
<u>Gate-Level Only Flow</u>	149
<u>Sample Script Starting from a Gate-Level Netlist</u>	149
<u>Sample Script Starting from a Timing-Optimized Netlist</u>	149
<u>LPS-DFT flow</u>	149
<u>Summary</u>	151

A

<u>More about BuildGates Synthesis</u>	153
<u>More about Optimizing the Design</u>	154
<u>Top-Down Optimization</u>	154
<u>Bottom-Up Optimization</u>	154
<u>More on do_optimize</u>	154

Cadence Synthesis Rapid Adoption Kit

<u>More about Timing Reports</u>	156
--	-----

B

<u>More about Datapath</u>	157
----------------------------------	-----

<u>More about Operator Merging</u>	158
--	-----

<u>More about Automatic Architecture Selection</u>	161
--	-----

<u>More about the AmbitWare Library</u>	163
---	-----

<u>Inference Verses Instantiation</u>	165
---	-----

<u>More about Extended Languages</u>	166
--	-----

C

<u>More about Low Power Synthesis</u>	169
---	-----

<u>More about LPS Clock Gating Technique</u>	170
--	-----

<u>More about TCF Files</u>	173
-----------------------------------	-----

<u>More about Gate-Level Power Optimization</u>	174
---	-----

<u>Gate Sizing</u>	174
--------------------------	-----

<u>Pin Swapping</u>	176
---------------------------	-----

<u>Buffer Removal</u>	176
-----------------------------	-----

<u>Gate Merging</u>	177
---------------------------	-----

<u>Slew Optimization</u>	178
--------------------------------	-----

<u>Logic Restructuring</u>	179
----------------------------------	-----

Cadence Synthesis Rapid Adoption Kit

Preface

This preface contains the following sections:

- [About This Manual](#) on page 7
- [Other Information Sources](#) on page 7
- [Documentation Conventions](#) on page 8

About This Manual

This manual describes how to use the BuildGates® Synthesis tool and the BuildGates® Extreme Synthesis tool.

Other Information Sources

For more information about BuildGates Extreme and other related products, you can consult the sources listed here.

- [*BuildGates Synthesis User Guide*](#)
- [*Command Reference for BuildGates® Synthesis and Cadence® PKS*](#)
- [*\[Datapath for BuildGates Synthesis and Cadence PKS\]\(#\)*](#)
- [*Design for Test \(DFT\) Using BuildGates Synthesis and Cadence PKS*](#)
- [*Distributed Processing of BuildGates® Synthesis*](#)
- [*HDL Modeling for BuildGates® Synthesis*](#)
- [*\[Low Power for BuildGates® Synthesis and Cadence® PKS\]\(#\)*](#)
- [*SDC Constraints Support Guide*](#)
- [*Common Timing Engine \(CTE\) User Guide.*](#)

Cadence Synthesis Rapid Adoption Kit

Preface

The following books are helpful references.

- IEEE 1364 Verilog HDL LRM
- Tcl Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company

Documentation Conventions

Text Command Syntax

The list below describes the syntax conventions used for the BuildGates Synthesis text interface commands.

<code>literal</code>	Non-italic words indicate keywords that you must enter literally. These keywords represent command or option names.
<code>argument</code>	Words in italics indicate user-defined arguments or information for which you must substitute a name or a value.
	Vertical bars (OR-bars) separate possible choices for a single argument.
[]	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one.
{ }	Braces are used to indicate that a choice is required from the list of arguments separated by OR-bars. You must choose one from the list.
	<code>{ argument1 argument2 argument3 }</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.
#	The pound sign precedes comments in command files.

Cadence Synthesis Rapid Adoption Kit

Preface

Using Menus

GUI commands can take one of three forms.

<i>CommandName</i>	A command name with no dots or arrow executes immediately.
<i>CommandName...</i>	A command name with three dots displays a form for choosing options.
<i>CommandName -></i>	A command name with a right arrow displays an additional menu with more commands. Multiple layers of menus and commands are presented in what are called command sequences, for example: File – Import – LEF. In this example, you go to the File menu, then the Import submenu, and, finally, the LEF command.

Using Forms

...	A menu button that contains only three dots provides browsing capability. When you select the browse button, a list of choices appears.
OK	The <i>OK</i> button executes the command and closes the form.
Cancel	The <i>Cancel</i> button cancels the command and closes the form.
Apply	The <i>Apply</i> button executes the command but does not close the form.

Cadence Synthesis Rapid Adoption Kit

Preface

Before You Begin

This Rapid Adoption Kit (RAK) consists of the following modules:

- [Getting Started with BuildGates Synthesis](#)
- [Getting Started with Design For Test](#)
- [Getting Started with Datapath](#)
- [Getting Started with Low Power Synthesis](#)

Each module highlights a specific feature for BuildGates[®] Synthesis.

Invoking BuildGates Synthesis or BuildGates Extreme Synthesis

For the first two modules you can use BuildGates[®] Synthesis. For the remaining modules you need to use BuildGates[®] Extreme Synthesis.

- To invoke BuildGates Synthesis in a text-based, Tcl shell, enter the following command at the UNIX prompt:

```
bg_shell
```

- To invoke BuildGates Synthesis in graphical user interface (GUI) mode, enter the following command at the UNIX prompt:

```
bg_shell -gui
```

Note: To invoke BuildGates[®] Extreme Synthesis, replace `bg_shell` with `bgx_shell`.

For help with your BuildGates Synthesis installation, see the [BuildGates Synthesis User Guide](#).

Command Language Interface

BuildGates Synthesis is one of the first EDA tools to use Tcl as its command language. Tcl was developed for EDA tools at UC Berkeley and has quickly become a command interface standard for EDA tools. Tcl is a simple language with very few rules, so it is easy to learn. For more details on the Tcl language, see either of these references:

- *Tcl and Tk Toolkit*, John Ousterhout, Addison-Wesley Publishing Company
- *Practical Programming in Tcl and Tk*, Brent B. Welch

Interactive commands entered into `bg_shell` (or `bgx_shell`) are saved in a command file called `ac_shell.cmd`. Commands and console messages are saved in a file `ac_shell.log`. You can use `ac_shell.cmd` input to `bg_shell` using the `source` command to recreate the previous execution of `bg_shell`. A setup file is saved in your directory under `.ambit/ambit.state` which saves your GUI color settings, default variables, and so on.

Getting Started

Set up your UNIX Environment

Make sure that your `.cshrc` file specifies the path to the directory in which the software is installed.

Download the Tutorial Data

1. Download the tutorial data to a new directory of your choice. For example,

```
cp -rf your_install_path/doc/syntut/RAK/RAK.tar.gz  
my_tutorial/.
```

2. Extract the data from the tar file:

```
cd my_tutorial  
gzip -dc RAK.tar.gz|tar -xvf -
```

This command creates the `RAK` directory in the current directory.

Cadence Synthesis Rapid Adoption Kit

Before You Begin

Install the Libraries

All modules share the same Artisan libraries.

To install the library, do the following:

1. Change directory to the `RAK` directory.
2. Enter the following command at the UNIX prompt:

```
INSTALL_LIB
```

The library installation process starts and you are asked whether or not you accept the Artisan license agreement.

Important

Read the agreement on terms of use. The agreement is also available in `LIB_LICENSE.TXT` (in the `my_tutorial/RAK` directory) for your review. By installing and using the library with this tutorial, you are agreeing to the terms of the agreement. If you do not agree to the terms as specified, we advise that you do not proceed with this tutorial.

Note: If you do not have `perl` installed on your system, you can unzip the `library.zip` file manually. Doing so indicates acceptance of the Artisan library license agreement as described in `LIB_LICENSE.TXT` file. For more information, read the `FAQ.txt` file.

Regarding the Results

Important

Your results might differ from the results shown in the different modules due to the difference in runs and release versions.

Cadence Synthesis Rapid Adoption Kit

Before You Begin

Getting Started with BuildGates Synthesis

- [Introduction to BuildGates Synthesis](#) on page 16
- [Objectives](#) on page 16
- [Example Design](#) on page 16
- [Setting Up to Run the Tutorial](#) on page 18
- [Basic BuildGates Flow](#) on page 19
 - [Setting Up Your Design Environment](#) on page 20
 - [Setting Global Variables](#) on page 20
 - [Reading the Technology Library](#) on page 21
 - [Reading the HDL Models for the Design](#) on page 23
 - [Creating a Generic Gate-Level Netlist](#) on page 24
 - [Setting Test Synthesis Assertions](#) on page 26
 - [Setting Constraints](#) on page 30
 - [Checking Constraints](#) on page 39
 - [Setting Path Groups](#) on page 42
 - [Optimizing the Design](#) on page 43
 - [Generating Reports](#) on page 45
 - [Writing Netlist and Database Files](#) on page 51
 - [Setting Scan Chain Assertions](#) on page 52
 - [Incremental Timing Optimization](#) on page 55
- [Interfacing with Physical Design Tools](#) on page 59
- [Using BuildGates Synthesis Graphical User Interface](#) on page 61
- [Migrating to BuildGates from Design Compiler](#) on page 66

- [Distributed Synthesis](#) on page 71
- [Summary](#) on page 72

Introduction to BuildGates Synthesis

BuildGates[®] Synthesis is a high-capacity and high-performance chip-level synthesis tool with built in signoff-quality, fast, full-chip timing engine. Fast and flexible, BuildGates Synthesis supports a wide variety of design styles such as multiple clocks, including both edge triggered and level sensitive with cycle stealing.

BuildGates Synthesis has a high capacity database that allows you to synthesize more of the design at once. Its fast runtime assures rapid turnaround, making chip-level synthesis practical. In addition, high-capacity enables productivity gains by eliminating the need for excessive resources and time required for elaborate bottom-up script development.

BuildGates Synthesis also offers automatic time budgeting, integration with physical design tools, VHDL and Verilog support, support of both reads and writes of netlist EDIF 2.0, Tcl command line interface for shell level control, transforms for performing focused optimizations, schematic and textual report capabilities, and integrated DFT analysis and scan insertion.

Capable of running in both command line mode and in graphical user interface (GUI) mode, the BuildGates Synthesis tool delivers dramatic performance and productivity benefits over conventional synthesis tools.

Objectives

In this brief self-teaching tutorial, you will become familiar with the BuildGates synthesis flow by running the tool on the test design. At the end of this module, you will have an understanding of BuildGates Synthesis methods, timing analysis, optimization methods, and test support.

Note: This module takes about an hour to run.

Example Design

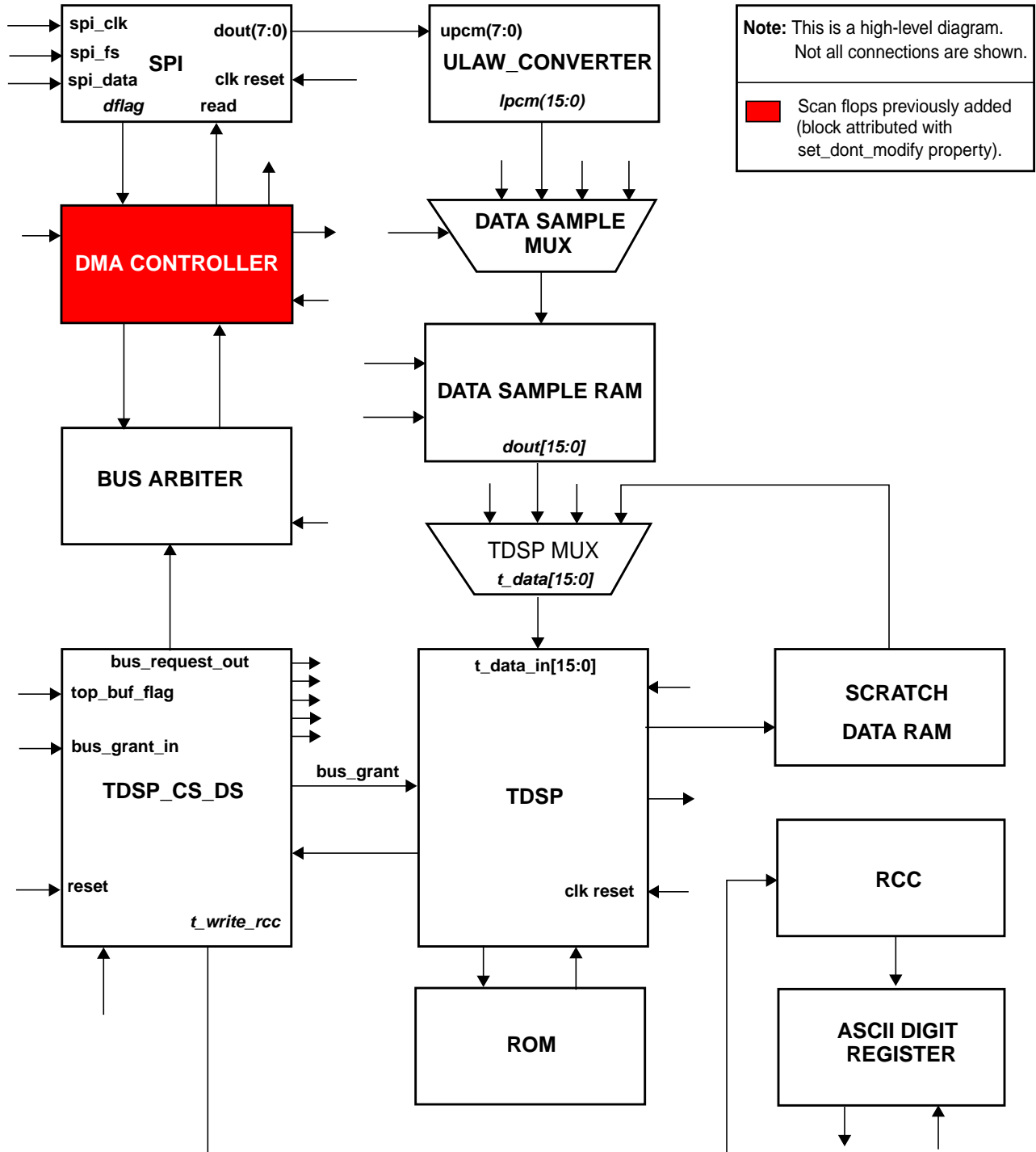
The example design provided with this module is a Verilog description of a Dual Tone Multi-Frequency (DTMF) receiver. In a telephone network, DTMF is a common in-band signaling technique used for transmitting information between network entities. DTMF signals are commonly generated by touch-tone telephones.

[Figure 2-1](#) on page 17 shows a block diagram of the DTMF receiver.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Figure 2-1 DTMF Block Diagram



Setting Up to Run the Tutorial

Note: We assume that you already copied the data. If not, see [Getting Started](#) on page 12.

Required License

You need a BuildGates (BG) license to run this module.

Required Files

You need the following files to run this tutorial:

- Synthesis library
- Verilog or VHDL netlist, preferably at RTL level.
- Constraints

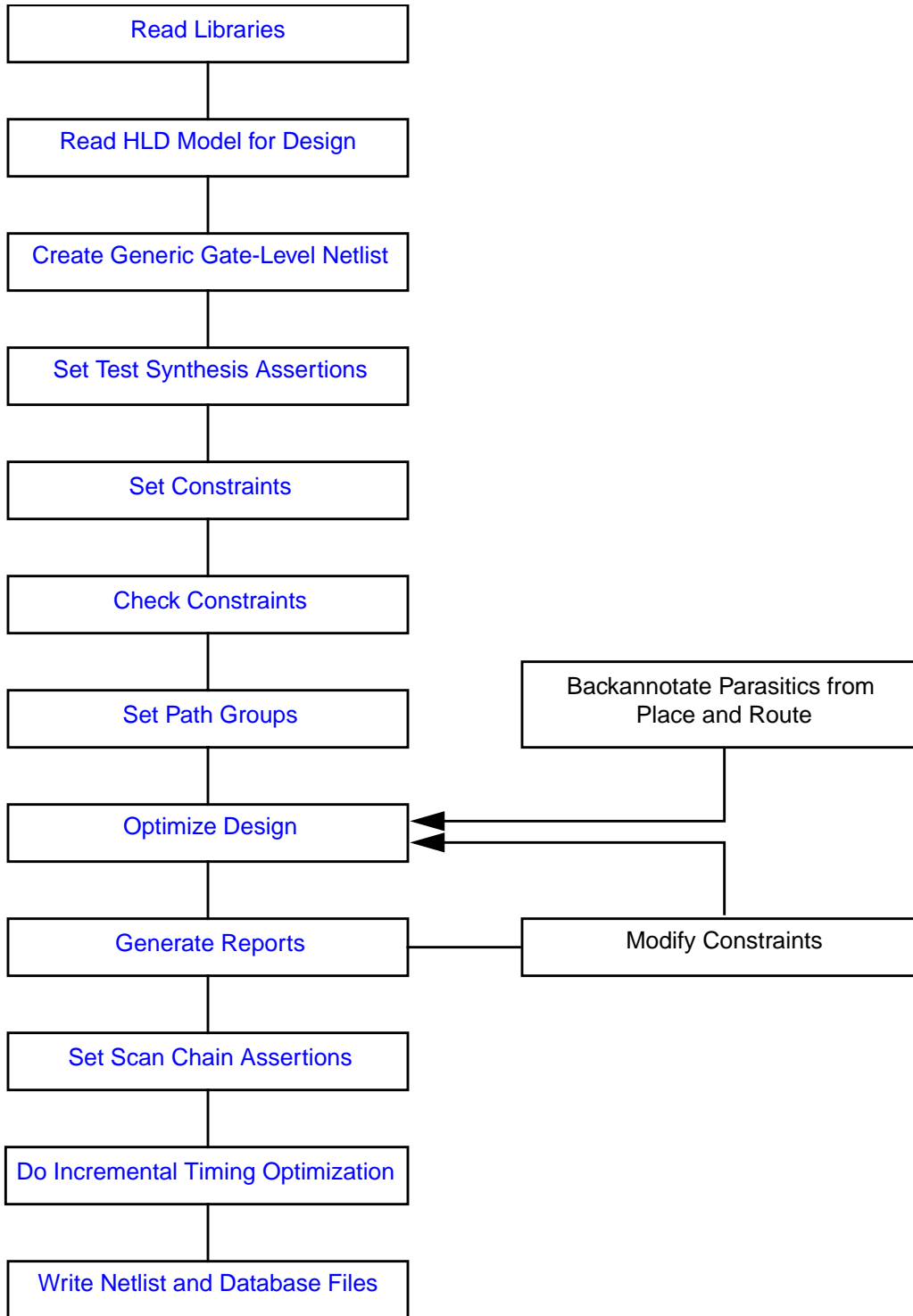
The `your_install_path/doc/syntut/RAK/tutorial_buildgates` directory contains the files needed to run this module. It has the following directory structure:

<code>tutorial_buildgates</code>	Root directory
<code>tcl</code>	Scripts to run this module
<code>rtl</code>	Verilog and VHDL design modules
<code>lib</code>	Synthesis libraries
<code>rpt</code>	Timing, area, hierarchy and other report files
<code>adb</code>	Database files
<code>run_dir</code>	Running directory

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Basic BuildGates Flow



Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Setting Up Your Design Environment

1. Change to the `run_dir` directory in the `tutorial_buildgates` directory.
2. For this tutorial, invoke BuildGates Synthesis in shell mode:

```
unix_shell> bg_shell
```

Note: You will explore the graphical user interface later on.

3. Set up your design environment by running the following script in `bg_shell`:

```
bg_shell> source ../tcl/setup.tcl
```

The `setup.tcl` file defines Tcl variables for the rtl, library, report, database, and Tcl directories. This file contains the following commands:

```
set rak_dir ..
set tcl_dir $rak_dir/tcl
set rtl_dir $rak_dir/rtl
set lib_dir $rak_dir/lib
set rep_dir $rak_dir/rpt
set adb_dir $rak_dir/adb
```

The variable `rak_dir` is set to the parent directory and other variables are set to their respective directories under the current directory.

Setting Global Variables

BuildGates Synthesis provides many global variables to control the overall run procedure and policies independent of the design. All global variables have default values when you start `bg_shell`. For more information on global variables, see the [*Global Commands for BuildGates Synthesis and Cadence PKS*](#).

You can set the value of any global variable by issuing the following command:

```
bg_shell> set_global var_name value
```

You can obtain the value of any global variable by issuing the following command:

```
bg_shell> get_global var_name
```

- Set some globals by running the following script:

```
bg_shell>source $tcl_dir/set_globals.tcl
```

The `set_globals.tcl` script sets three variables:

```
set_global message_verbosity_level 8
set_global echo_commands true
set_global report_precision 5
set_global fix_multiport_nets true
set_global sdc_write_unambiguous_names false
set_global line_length 1000
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- The `message_verbosity_level` variable controls the level of verbosity of the messages generated by `bg_shell` in reporting information, warnings, and errors.
- The `echo_commands` variable causes each command to be echoed in the standard output prior to its execution.
- The `report_precision` variable controls the number of digits appearing after the decimal point in timing reports, `report_timing` and `report_cell_instance_timing`. The default is 2.
- The `fix_multiport_nets` variable is considered during the `do_optimize` stage. When set to `true`, the `do_xform_fix_multiport_nets` command is run automatically and any output signal driving multiple output ports is split into different nets by inserting buffers in the paths.
- The `sdc_write_unambiguous_names` variable suppresses the output of unambiguous hierarchical names in sdc constraint.
- The `line_length` variable controls the line length in the console window.

Reading the Technology Library

A technology library containing timing and power models is required for successful mapping of RTL operations into technology-specific cells. In this tutorial, you will use the “Artisan TSMC 0.18u” libraries to optimize the design.

1. Read in the synthesis technology libraries by running the following script:

```
bg_shell> source $tcl_dir/read_lib.tcl
```

The `read_lib.tcl` script contains the following commands:

```
read_tlf $lib_dir/slow_4.3.tlf
read_tlf $lib_dir/p1lclk_slow_4.3.tlf
read_tlf $lib_dir/ram_128x16A_slow_4.3.tlf
read_tlf $lib_dir/ram_256x16A_slow_4.3.tlf
read_tlf $lib_dir/rom_512x16A_slow_4.3.tlf
read_library_update $lib_dir/tpz973gwc-lite_4.3.tlf
```

```
set_global target_technology slow
```

- The first five commands read in `.tlf` libraries. Although BuildGates currently supports TLF version 4.4, version 4.3 is used for this design.

You can load multiple `.alf`, and `.tlf` libraries in BuildGates with separate `read_alf` and `read_tlf` commands for each library.

- Files with the `.tlf` extension are in Cadence Timing Library Format.
- Files with the `.alf` extension are in Ambit Library Format.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Both `.alf` and `.tlf` formats are compatible with the Synopsys `.lib` format. Usually the ASIC or library vendors provide libraries in `.alf` and/or `.tlf` format.

If the vendor only provides a `.lib` file, you can compile the `.lib` file into a `.tlf` file using the `syn2tlf` utility (installed in the `your_install_dir/tools.platform/tlfUtil/bin` directory):

```
unix_prompt> syn2tlf my_cells.lib my_cells.tlf
```

In addition to the `.lib`, `.alf`, and `.tlf` formats, BuildGates Synthesis also supports Synopsys Stamp Models, IEEE 1481 Delay and Power Calculation system (DCL) libraries, Open Library API (OLA) libraries, and the GrayBox model for reusable megacells or IP cores.

For more information on supported library formats (and the `syn2tlf` utility), see the [Using Timing Libraries](#) chapter of the *Common Timing Engine (CTE) User Guide*.

- The `read_library_update` command merges pad cells from the `tpz973gwc-lite_4.3.tlf` pad library into the target technology library, `slow.alf`.

You can set the target technology with the `set_global target_technology` command. By default, the first library you read becomes the target technology. In addition to merging cells from another library, the `read_library_update` command can also be used for merging wireload models and operating conditions into the target library.

The previous example shows a complex use of the `read_library` command. However, in most cases, you can read the library with a single line:

```
read_tlf my.tlf
```

- The `set_global target_technology slow` command is used to specify the target library `slow.alf`. During optimization, BuildGates Synthesis maps a generic netlist to the cells from the target library `slow.alf`. You can set multiple target libraries by providing a list of libraries with the `set_global` command.

2. View the contents of the slow technology library by entering the following command:

```
bg_shell>report_library
```

The `report_library` command reports on the cells in the library, the wireload models, and the operating conditions.

Reading the HDL Models for the Design

BuildGates Synthesis supports Verilog, VHDL, and EDIF hardware description language (HDL) models. You can describe your design in the Verilog, VHDL, or EDIF language.

BuildGates Synthesis is compatible with the Synopsys synthesis tool. You can read the same RTL that you enter into the Synopsys tool into BuildGates Synthesis without making any changes. BuildGates even supports most of the Synopsys pragmas, such as `translate_off`, `translate_on`, and synthesis directives such as `full_case`, `parallel_case`, `infer_mux`. For more details on supported pragmas and synthesis directives, see [Supported Synopsys Directives](#) in the *HDL Modeling for BuildGates Synthesis*.

- Read in the HDL model for the design by entering the following command:

```
bg_shell>source $tcl_dir/read_rtl.tcl
```

The `read_rtl.tcl` script contains the following commands:

```
set_global hdl_verilog_vpp_arg -I$rtl_dir
read_verilog $rtl_dir/accum_stat.v
read_verilog $rtl_dir/alu_32.v
read_verilog $rtl_dir/arb.v
read_verilog $rtl_dir/data_bus_mach.v
read_verilog $rtl_dir/data_sample_mux.v
read_verilog $rtl_dir/decode_i.v
read_verilog $rtl_dir/digit_reg.v
read_verilog $rtl_dir/dma.v
read_verilog $rtl_dir/dtmf_recvr_core.v
read_verilog $rtl_dir/execute_i.v
read_verilog $rtl_dir/mult_32_dp.v
read_verilog $rtl_dir/ml6x16.v
read_verilog $rtl_dir/port_bus_mach.v
read_verilog $rtl_dir/prog_bus_mach.v
read_verilog $rtl_dir/ram_128x16_test.v
read_verilog $rtl_dir/ram_256x16_test.v
read_verilog $rtl_dir/results_conv.v
read_verilog $rtl_dir/spi.v
read_verilog $rtl_dir/tdsp_core.v
read_verilog $rtl_dir/tdsp_core_glue.v
read_verilog $rtl_dir/tdsp_core_mach.v
read_verilog $rtl_dir/tdsp_data_mux.v
read_verilog $rtl_dir/tdsp_ds_cs.v
read_verilog $rtl_dir/test_control.v
read_verilog $rtl_dir/ulaw_lin_conv.v
read_verilog $rtl_dir/iopads.v
read_verilog $rtl_dir/dtmf_chip.v
```

The `set_global hdl_verilog_vpp_arg -I$rtl_dir` command enables the Verilog pre-processor to search Verilog files in the `$rtl_dir` directory.

You can see that you can read multiple Verilog files using the `read_verilog` command for each file. The `read_verilog $rtl_dir/accum_stat.v` command reads the `accum_stat.v` file from the `rtl_dir` directory.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

To read a VHDL file, you can use the `read_vhdl` command. You can use the `set_vhdl_library_logical_library_name_directory` command to associate the logical library with a physical directory. You can set the `hdl_vhdl_environment` variable to `synopsys` to select the arithmetic package supported by the Synopsys VHDL compiler. By default, this variable is set to `standard` which is the IEEE standard package.

While parsing the design modules, the `read_verilog` and `read_vhdl` commands will report syntax errors, warnings, and other problems. For example, one of the warnings you see while parsing the `accum_stat.v` file is:

```
--> WARNING: 'delay' statement is not supported for synthesizable modules (File
./rtl/accum_stat.v, Line 68) <VLOGPT-035>.
assign #1 acc_v = accum['S_ACC'] ;
```

`#1` specifies a one-unit delay in Verilog. The `#` construct is typically used for simulation, but it is not a synthesizable construct. You can turn off the above warning message by using the command, `set_message_verbosity VLOGPT-035 off`.

Creating a Generic Gate-Level Netlist

- Transform the design into a hierarchical, gate-level netlist by entering the following command:

```
bg_shell>do_build_generic
```

The `do_build_generic` command transforms the design into a hierarchical, gate-level netlist consisting of technology-independent logic gates using:

- Components from the Ambit Synthesis Technology Library (ATL)
- Arithmetic and logical operators from the AmbitWare Components Library (AWACL)

The command performs constant propagation, loop unrolling, lifetime analysis, register inferences, and logic mapping. You must execute this command before any optimization commands (such as `do_optimize` or `do_xform_*`) can be applied.

Note: If you are reading a gate-level netlist mapped to cells from the library, you still need to run the `do_build_generic` command.

For more information about the generic synthesis step, see the *BuildGates Synthesis User Guide*. By default, the `do_build_generic` command uses timing sensitive implementation for all the operators. BuildGates Synthesis also provides an architecture selection directive to select different types of architectures for arithmetic operators. In BuildGates Extreme Synthesis, architectures are automatically selected based on timing and context. For more information, see [Chapter 4, "Getting Started with Datapath."](#)

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The following is one of the messages generated by the `do_build_generic` command:

```
Info:      Processing design 'spi' <CDFG-303>.
          Statistics for case statements in module 'spi' (File ./rtl/spi.v)
          <CDFG-800>.
          +-----+
          | Case Statistics Table |
          +-----+
          | Line | Type | Full | Parallel |
          +-----+
          |  97 | case | AUTO |   AUTO   |
          +-----+
```

This message provides information on a `case` statement at line 97 in the `spi` module. The case statement type can be `case`, `casex`, or `casez`. It tells you whether the case statement can be implemented with the full case (priority encoding logic) or parallel case (parallel encoding logic) synthesis directive. With the `AUTO` option, the tool will decide to implement the case statement using either full case or parallel case depending on the constraints. If the synthesis directives in the RTL include either `full` or `parallel`, the `Full` and `Parallel` options in the case statistics table will be set to `YES`. If the full case synthesis directive is incorrectly used, the `do_build_generic` command generates warnings for possible RTL and gate-level simulation mismatches.

Each sequential device that is inferred by the `do_build_generic` command is reported in a sequential element table as shown below.

```
+-----+
| Sequential Elements |
+-----+
| Module | File | Line | Name | Type | Width | AS | AR | SS | SR |
+-----+
| tdsp_ds_cs | ./rtl/tdsp_ds_cs.v | 128 | t_sel_7_reg | FF | 1 | N | Y | N | N |
| tdsp_ds_cs | ./rtl/tdsp_ds_cs.v | 141 | t_bit_7_reg | FF | 1 | N | Y | N | N |
+-----+
Finished processing module: 'tdsp_ds_cs' <MODGEN-110>.
```

The first line of data in this table indicates that line 128 in the `tdsp_ds_cs.v` file infers a register named `t_sel_7_reg`. It is of type `D_FF` and has 1 bit width. This register has no (N) asynch set control (AS), synch set control (SS), synch reset control (SR), but does have (Y) asynch reset control (AR).

For more information on case statements and RTL coding styles for synthesis, see the [HDL Modeling for BuildGates Synthesis](#).

Checking the Netlist

After you create a generic netlist using the `do_build_generic` command, Cadence recommends that you check the structural connectivity of the netlist.

1. To check the structural connectivity of the netlist, enter the following command:

```
bg_shell>check_netlist
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The `check_netlist` command performs a number of checks on the structural connectivity of the netlist, including checks for recursively defined modules, combinational feedback, undriven nets and pins, multiply driven nets and pins, and undriven ports. You can run the `check_netlist` command on a mapped netlist as well.

The `check_netlist` command generates the following report:

```
Current module: dtmf_chip
The number of recursively defined modules(at least):      0
The number of combinational feedbacks:                   0
The total number of undriven nets/pins:                   1
The total number of multiply driven nets:                 0
The total number of potential multiply driven nets:      0
The total number of undriven ports:                       0
The total number of common inputs between data and enable: 0
```

You should justify all warnings before proceeding further.

2. To see the details of the undriven nets and pins, use the `-verbose` option:

```
bg_shell>check_netlist -verbose
```

This command displays warnings like this:

```
--> WARNING: Undriven instance pin 'DTMF_INST/int' found in module 'dtmf_chip'
          <FNP-533>.
```

In this example, you can ignore the warning because the `int` pin in instance `DTMF_INST` is not used.

Setting Test Synthesis Assertions

As ASICs become more complex, it has become a standard practice to use test synthesis during logic synthesis to integrate design-for-test (DFT) logic into designs. The foundry's tester uses these DFT logic structures to check the chip for manufacturing defects. The logic structures produced by test synthesis do not affect the function of the chip.

The test synthesis capabilities of BuildGates Synthesis perform a single-pass, automatic, full-scan insertion prior to and during logic optimization. The test synthesis capability allows you to trade off timing and area during optimization. Currently, it does not perform automatic test pattern generation (ATPG), boundary scan/JTAG, and BIST. BuildGates Synthesis is designed to work well with the FastScan[®] tool from Mentor Graphics and the Turboscan[®] tool from Syntest, and it is compatible with most other vendors as well. For more information about BuildGates test features, refer to [Chapter 3, "Getting Started with Design For Test."](#)

Finding Objects

To set assertions and constraints, you must be able to find various objects such as ports, net names, instance names and module names in your design database.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The `find` command finds design objects and prepares a list for other `bg_shell` commands. Unless you specify the `-regexp` option, BuildGates uses glob-style pattern matching to find design objects in the design database. Multiple wildcard matching in a name is permitted.

BuildGates Synthesis assigns a unique identifier (ID) to each object type (cell, component, module, instance, net, pin, port, and so on). The `find` command returns the object ID, which is an integer number. To get more information on an object, use the `get_info` command.

Typically, the output of the `find` command (a list of objects found by this command) is an argument to another command that accepts a list of objects and performs some task.

For example, the following command sets all input ports of a module to the same input delay:

```
set_input_delay 0.2 [find -ports -input *] -clock master_ck
```

Setting Assertions

1. Set assertions for test modes and connection preferences by running the following script:

```
bg_shell>source $tcl_dir/scan_assert.tcl
```

The `scan_assert.tcl` script contains the following commands:

```
issue_message -type info "Mapping test_control module"

set_current_module test_control
do_xform_map
set_dont_modify [find -hier -module test_control]
set_current_module [find -module dtmf_chip]

issue_message -type info "Setting up for scan synthesis in tieback mode"

set_scan_style muxscan
set_global dft_scan_path_connect tieback
set_global dft_scan_avoid_control_buffering true
set_scan_mode IOPADS_INST/Pscanenip/C 1

check_dft_rules
```

The script contains two sections.

2. Examine the commands in the first section of the `scan_assert.tcl` file.

This section applies to the `test_control` module only. This module consists of a scan clock muxed with all the clocks used in the `dtmf_chip` module and is controlled by the `test_mode` signal. For this tutorial, you do not want to optimize this module.

- The `set_current_module` command sets `test_control` as the design context for DFT, then sets the context back to the DTMF design for optimization. You will look at this command in more detail in [Setting Constraints](#) on page 30.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- ❑ The `do_xform_map` transform maps the `test_control` module to target technology `slow`. In this case, all you are doing is mapping the test control logic to the target technology, and then telling the tool not to modify it. Since this is test control logic, it is not speed sensitive. [Optimizing the Design](#) on page 43 describes the transform commands in more detail.
- ❑ The `set_dont_modify` command keeps the instances, nets, or submodules in the `test_control` module from being modified by the optimization tool.
- ❑ The second `set_current_module` command sets the timing and optimization context to the `dtmf_chip` module for optimization.

3. Examine the commands in the second section of the `scan_assert.tcl` file.

- ❑ The `set_scan_style` command sets the scan style to `muxscan` (muxed-D scan flops) for the current module and all child modules. BuildGates also supports scan flops of type `clocked_scan`, `clocked_lssd`, and `aux_clocked_lssd`.
- ❑ The `set_global dft_scan_path_connect tieback` variable sets the scan mode to `tieback` for later connection. The `tieback` mode connects the scan-data output pin of the scan cell back to its own scan-data input pin. Cadence recommends that you set the scan mode to `tieback` before optimization. After the scan chain is created and connected, you can set the value of this global variable to `chain`.
- ❑ The `set_global dft_scan_avoid_control_buffering true` command is optional. It prevents buffering of the high fanout scan mode control signal after it is inserted. By default, high fanout signals are buffered by timing optimization and design rule fixing. This command overrides the default behavior for the scan mode signal only.
- ❑ The `set_scan_mode IOPADS_INST_Pscanenip/C 1` command specifies that input port `IOPADS_INST_Pscanenip/C` activates shifting of scan data through the scan chain when the port is active high (1). When you use the `set_scan_mode` command to define the input port of a scan chain, you must specify whether the scan chain is active-high (1) or active-low (0). There is no default for this argument. If you do not use the `set_scan_mode` command, the test synthesis tool creates an active high top-level port with the default name `BG_scan_enable`.
- ❑ The `check_dft_rules` command checks your design for selected DFT rule violations such as uncontrollable clocks and uncontrollable asynchronous signals.

The tool does not insert scannable elements where DFT violations occur. It displays a message if it detects any violations. You should fix all reported problems either by modifying your RTL or by using the auto-DFT fix capabilities described in [Improving Testability of Your Design](#) in *Design For Test (DFT) Using BuildGates Synthesis and Cadence PKS*. Otherwise, your fault coverage is reduced.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

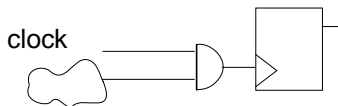
The following error message is one of the messages reported by `check_dft_rules`:

```
==> ERROR: DFT Violation 6: Internally driven Clock net 'DTMF_INST/
m_digit_clk' in module 'dtmf_recvr_core' (File ./rtl/dtmf_recvr_core.v, Line
79) <DFT-316>.
      Traced its effective fanin cone to:
          :test_mode
          scan_clk
          DTMF_INST/RESULTS_CONV_INSTdigit_clk_reg:Q
```

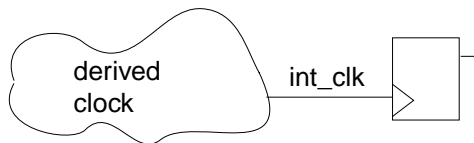
In the DTMF design, all clocks are derived from the primary clock `refclk` using a Phase Lock Loop (PLL), which causes a DFT violation. In muxed scan style, all clocks must be controllable from a primary input to allow the tester to control the registers during the test process. Figure 2-2 shows some examples of DFT violations.

Figure 2-2 Examples of DFT Violations

DFT Violation: Gated Clock



DFT Violation: Derived Clock



Fixing DFT Violations

You can fix DFT violations by fixing the RTL or using the auto-fix DFT capabilities in the tool. Additionally, if test mode activated logic has been added to the RTL, DFT violations specific to this test mode logic, can be safely bypassed by using the `set_test_mode_setup` assertion. You can fix the previous DFT violation by using the `test_mode` signal to bypass the clock gating logic or internal clocks during test mode.

- Fix the violations by running the following script:

```
bg_shell>source $tcl_dir/fix_dft_rules.tcl
```

The `fix_dft_rules.tcl` script contains the following commands:

```
set_test_mode_setup test_mode 1
set_test_mode_setup reset 0
check_dft_rules
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- The `set test mode setup test_mode 1` command sets the `test_mode` input port to 1 during the entire test session. The second `set_test_mode_setup` command sets the `reset` port to 0 for the entire test session.
- The `check_dft_rules` command displays a summary of the DFT clock domains, any internal clock domain points (specified using the `set_dft_internal_clock_domain` assertion), and the number of registers associated with those domains that have passed the DFT rules checks:

```
Info:      Partitioning registers for scan based on clock domain. <DFT-325>.
Clock Domain 0 from pin 'scan_clk' (Neg Edge) <Internal Pin: 'None'> has 129 f/f
Clock Domain 1 from pin 'scan_clk' (Pos Edge) <Internal Pin: 'None'> has 411 f/f
Total Clock domains: 2 for 540 f/f
Info:      Total Scannable register count: 540 <DFT-340>.
```

In addition to uncontrollable clocks, internally generated clocks, and uncontrollable asynchronous signals, you can check your design for combinational loops and flip-flops with potential race-conditions between the data and clock signals by setting the following variables before you run `check_dft_rules`.

```
set_global dft_enable_combinational_loop_check true
set_global dft_enable_race_condition_check true
```

If you set `dft_enable_race_condition_check` for the DTMF design you will get the following warning message when you run `check_dft_rules`:

```
Info:      Checking for registers with clock/data race condition. <DFT-326>.
--> WARNING: Input 'test_mode' fans out to both data and enable pins of
sequential element 'DTMF_INST/SPI_INST/dflag_reg' of module 'spi.'
This may create potential race conditions. Try '-flatten on' on
this module <FNP-543>.
```

For more information about BuildGates Synthesis test features, such as ATPG, boundary scan, and scan chain re-ordering refer to [Chapter 3, "Getting Started with Design For Test."](#)

Setting Constraints

In addition to libraries and a functional RTL model of the design, the logic synthesis process requires a set of constraints on the design. By default, BuildGates Synthesis meets your timing constraints first, then tries to achieve the smallest area. For more information on changing the area and timing priority or making area a priority, see the [Common Timing Engine \(CTE\) User Guide](#).

The performance generally refers to the maximum clock frequency at which the design implementation (netlist) can operate. You do not need to specify any area constraints for the design because BuildGates always strives for the smallest possible design area for the given timing and physical constraints.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Important

BuildGates Synthesis always tries to meet the specified timing constraints. Do not synthesize your design without basic I/O and clock constraints because BuildGates is a performance, timing-driven synthesis engine. Do not over-constrain the design to meet the desired timing constraints. For example, you do not need to set an 8 ns clock to meet a 10 ns clock constraint, which is a common practice in other synthesis tools. Do not set your I/O arrival times at 95% of the clock period, as is common in other synthesis tools. If you over constrain the design, you will likely experience very long run times while the tool works very hard to meet these constraints.

Note: If you already have Synopsys Design Compiler (DC) or Prime Time (PT) timing constraints, use the BuildGates Synthesis Constraint Translator to translate these constraints to a BuildGates Synthesis constraints file. For more information, refer to [Migrating to BuildGates from Design Compiler](#) on page 66.

- Set the commonly used constraints by running the following script:

```
bg_shell>source $tcl_dir/constraints.tcl
```

The following sections discuss the different sections of the constraint file in detail:

- [Setting Up Hierarchical and Timing Context](#) on page 31
- [Setting Clock Constraints](#) on page 32
- [Setting Constraints on Primary I/O Ports](#) on page 36
- [Setting Multicycle and False Paths](#) on page 37
- [Setting Design Rule Constraints](#) on page 37
- [Setting Wire Load Models](#) on page 38

Setting Up Hierarchical and Timing Context

The Setting up Hierarchical and Timing Context section of the `constraints.tcl` file contains the following commands:

```
# Setting up Hierarchical and Timing Context"
issue_message -type info "--> Setting up Hierarchical and Timing Context ..."

set_current_module dtmf_chip
set_top_timing_module dtmf_chip
```

- The first command issues the message “Setting up Hierarchical and Timing Context.” This message is written in the log file.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- The `set_current_module` command makes `dtmf_chip` the current module. Design objects such as nets and ports, referenced by subsequent commands, must be specified relative to this module. All searches for design objects start in the `dtmf_chip` module. Hierarchical design objects are referenced with `dtmf_chip` as the root (top) of the hierarchy.

Note: If a module has multiple instances and each instance requires a separate set of constraints, you can use the `set_current_instance` command.

- The `set_top_timing_module` command identifies the module to be used by subsequent commands as a context for setting timing constraints. All constraints are set with reference to the `dtmf_chip` module. The optimization commands will operate on the `dtmf_chip` module and its hierarchy.

This is a unique and very powerful feature of BuildGates Synthesis. If you set a new top timing module, you change the context for the subsequent timing constraints and the optimization steps. The constraints applied to the previous top timing module are preserved but do not affect the steps carried out in the new top timing module.

Setting Clock Constraints

The DTMF design has a primary clock port, `refclk`, which goes through a phase locked loop circuit and internally generates a clock, `clk`, with a 6 ns period and several divided clocks `rcc_clk`, `spi_clk`, `dsram_clk`, `ram_clk`, and `digit_clk`, each with 12 ns periods. All these clocks are muxed with the scan clock in the `test_control` module. You will set the clock definition for these internal clocks at the output of the `test_control` module.

BuildGates uses the concept of ideal and applied clocks. An ideal clock is a perfect representation of the clock, while the applied clock is a clock applied to the system. For example, on a lab bench the ideal clock is the clock source (for example, waveform generator) and the applied clock is the connection to the chip under test. The advantage of using an ideal clock and applied clocks is that a common ideal clock can drive any number of applied clocks, all with different insertion delays while still maintaining the fundamental timing relationship. The split between ideal and applied makes it simple to construct complicated interrelated clock systems.

Clock constraints are required to set the timing constraints on combinational logic between registers.

1. Examine the commands defined in the Setting Ideal Clocks section of the `constraints.tcl` file.

```
issue_message -type info "--> Setting Ideal Clocks ..."  
set_clock vclk1 -period 6.0 -waveform { 0 3.0 }  
set_clock vclk2 -period 12.0 -waveform { 0 6.0 }
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- ❑ The first `set_clock` command defines the ideal clock `vclk1` with period 6, leading transition at 0, and trailing transition at 3: Since the leading transition is a rising edge, `vclk1` is treated as a positive clock.
- ❑ The second command defines another positive ideal clock, `vclk2`, with period 12, leading transition at 0, and trailing transition at 6:

These two ideal clocks will be used as a global reference signal for all the data signals in the DTMF design. In a single clock design, you will need only one ideal clock. However, in a multi-phase clock design, you will need several ideal clocks.

Once the ideal clocks are defined, you can define actual clock signals arriving at input/internal ports of the DTMF design.

2. Examine the commands defined in the Setting Primary Clocks section.

```
issue_message -type info "--> Setting Primary Clocks ..."  
set_clock_root -clock vclk1 [find -port refclk ]
```

The `set_clock_root` command assigns positive polarity to ideal clock `vclk1` and associates primary input pin `refclk` (system clock) with ideal clock `vclk1`.

3. Examine the commands defined in the Setting Internally Generated Clocks section.

```
issue_message -type info "--> Setting Internal Clocks ..."  
set_generated_clock -name vclk1_int1 -from \  
DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/clk \  
-divide_by 2 DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/write_reg/Q  
set_generated_clock -name vclk1_int2 -from DTMF_INST/DMA_INST/clk \  
-divide_by 2 DTMF_INST/DMA_INST/write_reg/Q
```

The `set_generated_clock` commands create two generated clocks `vclk1_int1` and `vclk1_int2`, which are derived from the clock source `vclk1`. This command creates a new clock signal from the clock waveform of a given pin in the design, or an existing virtual (ideal) clock signal, and binds it with the targeted pins (listed at the end of the command). Whenever the source clock changes, the derived clock(s) change automatically. You can generate the new clock waveform by

- ❑ Multiplying the frequency of the source clock
- ❑ Dividing the frequency of the source clock
- ❑ Selecting the edges of the source clock to be mapped on to the edges of the new clock

For examples about assertions on generated clocks, see [Specifying Generated Clocks](#) in the *Common Timing Engine (CTE) User Guide*.

4. Examine the commands defined in the Setting Muxed Clocks section.

```
issue_message -type info "--> Setting Muxed Clocks ..."  
set_clock_pin [find -hier -pin DTMF_INST/TEST_CONTROL_INST/m_clk]
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

```
set drive_pin [get_drive_pin -hier [get_info $clock_pin net] ]
set_clock_info_change -clock vclk1 -pos $drive_pin
set_clock_insertion_delay -pin $drive_pin 2.0

foreach clock {m_rcc_clk m_spi_clk m_dsram_clk m_ram_clk m_digit_clk} {
  set clock_pin [find -hier -pin DTMF_INST/TEST_CONTROL_INST/$clock]
  set drive_pin [get_drive_pin -hier [get_info $clock_pin net] ]
  set_clock_info_change -clock vclk2 -pos $drive_pin
  set_clock_insertion_delay -pin $drive_pin 2.0
}

set_clock_uncertainty 0.25
```

- ❑ The `set clock_pin` command defines the `clock_pin` Tcl variable and assigns it a value of `DTMF_INST/TEST_CONTROL_INST/m_clk`, which is a clock pin.
- ❑ The `set drive_pin` command defines the `drive_pin` Tcl variable and assigns it a value of the pin driving the `DTMF_INST/TEST_CONTROL_INST/m_clk` net.
- ❑ The `set_clock_info_change` command assigns the positive polarity (`-pos`, which is the default) to ideal clock `vclk1` and associates the instance pin driving the `DTMF_INST/TEST_CONTROL_INST/m_clk` net (actual clock) with the ideal clock.

As you can see, you can associate pins with more than one ideal clock.

You can use `set_clock_info_change` to change the clock to data, data to clock, and clock to clock for paths going through the specified pins for the downstream logic. This command is useful for modeling frequency dividers and clock-shaping circuits.

Note: In version 5.0 and beyond, BuildGates Synthesis automatically changes a clock network to a data network, so you do not need to use the `set_clock_info_change` command to change a clock signal to a data signal, but you still need to use it if you want to change a data signal to a clock signal.

- ❑ The `set_clock_insertion_delay` command inserts a 2-unit delay as clock network latency. This means it will take a 2-unit delay for the clock signal to propagate from the clock definition point (instance pin driving `DTMF_INST/TEST_CONTROL_INST/m_clk` net) to the clock pin of a register. “Clock network latency delay” is commonly referred to as “clock tree delay.”

The `set_clock_insertion_delay` command can also be used to add clock source latency, the time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point (clock port) in the design. You need to use the `-source` option to add clock source latency. In this design, you could use the `set_clock_insertion_delay -source` command to model the delay through a PLL circuit.

You can specify clock network latency and clock source latency on the clock waveform as well as the clock port or clock pin. Clock tree delay is used during ideal

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

clock propagation mode, which is a default mode. Clock tree delays set by the `set_clock_insertion_delay` command are ignored during clock propagation mode. Actual clock tree delays are used during propagated mode. The `set_clock_propagation` command tells the timing analyzer to treat all the clock signals associated with the current timing top module as either ideal or propagated.

- ❑ The `foreach Tcl` loop associates clock pins `m_rcc_clk`, `m_spi_clk`, `m_dsram_clk`, `m_ram_clk` and `m_digit_clk` on `DTMF_INST/TEST_CONTROL_INST` instance with ideal clock, `vclk2`. For this tutorial, you will not assign `clock_insertion_delay` for these clocks.

Clock pins, `m_rcc_clk`, `m_dsram_clk`, and `m_ram_clk` are internally derived from frequency divider logic implemented inside the `DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST` and `DTMF_INST/DMA_INST` modules. These clock signals are connected to Q pin `write_reg` registers. Hence, they will be treated as data signals. The two `set_clock_info_change` commands in the Setting Internally Generated Clocks section, are used to change data signals to clock signals for the paths going through the `write_reg/Q` pin for the downstream logic.

- ❑ The `set_clock_uncertainty` command specifies a 0.25 unit delay as a clock skew for the `vclk1` and `vclk2` clock domains. It can also be used to specify inter-clock skew for different clock domains. Clock uncertainty can also be used as a general method to add pessimism and/or optimism. A positive uncertainty contributes to pessimism. A negative uncertainty has no meaning physically, but it makes all slack numbers optimistic. Negative uncertainty can be used to adjust conservative constraints into more realistic ones. Figure 2-3 shows the skew between the clocks, `clk1` and `clk2`.

Figure 2-3 Clock Skew



For more information on clock commands or methods, refer to the [Common Timing Engine \(CTE\) User Guide](#).

Setting Constraints on Primary I/O Ports

You need to constrain the signal changes arriving at primary input ports of the DTMF design. These signal changes are controlled by an external block driving the DTMF block. To constrain the output ports, you need to consider the timing requirement of downstream modules to which output ports of the DTMF block are connected.

- ▶ Examine the commands defined in the Setting Input/Output ports constraints section of the `constraints.tcl` file.

```
issue_message -type info "--> Setting Input/Output ports constraints..."
set_input_delay -clock vclk1 0.5 [ get_names [ find -inputs -no_clocks] ]
set_external_delay -clock vclk1 0.5 [ get_names [ find -outputs port*] ]
set_external_delay -clock vclk1 0.5 [ get_names [ find -outputs tdigit*] ]
```

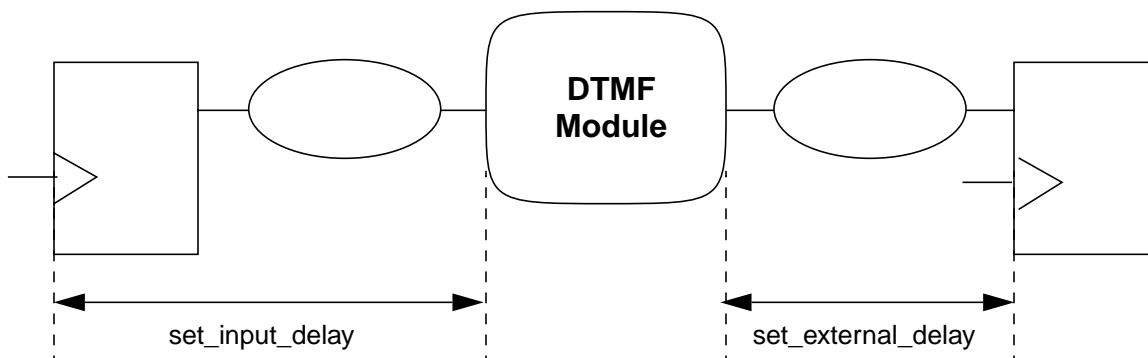
- The `set_input_delay` command (see Figure 2-4) sets a 0.5 unit delay for the signals to arrive at all input ports of the DTMF block.

The rising and falling edge transitions of data signals arrive at all input ports at a 0.5 unit delay after the leading edge of the `vclk1` clock. Since you have not set the `-early` or `-late` option, the input delay is applied for both late (max) and early (min) signals.

- The `set_external_delay` commands (see Figure 2-4) set external delays of 0.5 units on output ports whose names begin with `port` and `tdigit`. These ports are driven by registers clocked by `vclk1`.

The required time at the output port is computed as if the output port is driving a register external to the design. The data signal must arrive at the input of the external register (including setup time) before the arrival of the clock. In other words, the required time for the data signal at the output port of your design is calculated by subtracting the propagation delay through the logic from the clock arrival time at the register of the downstream block.

Figure 2-4 Input and External Delays



Setting Multicycle and False Paths

By default, all paths in a design are considered to be single-cycle paths. However, in the DTMF design, signal propagation for paths going through multipliers spans over multiple cycles. You need to relax the timing constraints on these paths.

- ▶ Examine the commands in the Setting Multicycle and False Paths section of the `constraints.tcl` file:

```
issue_message -type info "--> Setting Multicycle and False Paths ..."  
set_cycle_addition -to DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/acc_reg* 1  
set_cycle_addition -to DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/p_reg* 1  
set_cycle_addition -to DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/ov_flag_reg* 1  
  
set_false_path -from reset  
  
set_constant_for_timing 0 test_mode  
set_constant_for_timing 0 scan_en
```

- The `set_cycle_addition` commands indicate that all paths ending in pins `acc_reg*`, `p_reg*` and `ov_flag_reg*` registers in the `DTMF_INST/TDSP_CORE_INST/EXECUTE_INST` hierarchy should be treated as two-cycle paths—one cycle addition to the clock driving these registers.

The corresponding Synopsys command is `set_multicycle_path 2`. One of the most powerful and unique features of the `set_cycle_addition` command is that the cycle addition number can be a real number. For example, you could use the command, `set_cycle_addition -0.2`, to constrain paths for 80% of the clock period.

- The `set_false_path` command sets paths starting from the reset pin as false path. The timing analyzer ignores these paths, and the paths are optimized for area and not timing.
- The `set_constant_for_timing` commands set the value of the primary ports `test_mode` and `scan_en` to 0.

Logic 0 will be propagated through corresponding combinational logic cones and disable sections of the logic for timing analysis.

Setting Design Rule Constraints

In order for a circuit to function correctly, it needs to meet design rule constraints as well. You can set all the necessary design rule constraints such as fanout, port capacitance, slew time, wire loads, and operating conditions in BuildGates Synthesis.

- ▶ Examine the commands in the Setting Design Rules section of the `constraints.tcl` file.

```
issue_message -type info "--> Setting Design Rules ..."  
set_slew_time_limit 2.3 [ find -ports -noclocks * ]
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

```
set_global fanout_load_limit 15

set_drive_cell -cell PDO04CDG [find -input -noclocks * ]
set_port_capacitance [expr [get_cell_pin_load -cell PDIDGZ -pin PAD]*2.0]\
[find -output * ]
```

- The `set_slew_time_limit` command specifies the limit for slew time (transition time) at the input and output ports of the DTMF module.

You can also set global values for the slew time using the global variables `max_slew_time_limit` and `min_slew_time_limit`, and you can use the `set_slew_time_limit` command to override these global values.

- The `set_global fanout_load_limit` command specifies the maximum value for fanout load limit (number of transistors to be connected) on the ports of a cell and is used to enforce design rule checks.
- The `set_drive_cell` command models the drive strength of external drivers connected to input ports of the DTMF block.

This command identifies output pin PAD of the PDO04CDG library cell driving input ports of the DTMF block. You can also use the `set_drive_resistance` command to set the drive resistance.

- The `set_port_capacitance` command sets output loads by specifying the output port capacitance external to the DTMF block.

Setting Wire Load Models

For deep sub-micron designs, using wire load models for synthesizing is becoming less attractive because the margin of error in these designs is disappearing. Thus, wire load based design, or “design by guard band” is becoming less and less valid with each new technology generation. Designers are using tools like BuildGates Physically Knowledgeable Synthesis (PKS) to accurately predict interconnect RCs rather than to use wire load models. This next generation of tools does “design by accuracy.”

However, you can use BuildGates Synthesis to synthesize designs using wire load models.

- Examine the commands in the Setting wire load models section of the `constraints.tcl` file.

```
set_wire_load TSMC18_Conservative
set_wire_load_mode enclosed
set_wire_load_selection_table WireAreaCon
```

- The `set_wire_load` command sets the wire load model for the current module, DTMF. If you run the `report_library -wireload` command, you see `TSMC8k_Conservative` as the default. The `set_wire_load` command overrides this default.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- The `set_wire_load_mode` command specifies that the wire load model must be selected using an area lookup table for the module at the lowest level in the design hierarchy that contains the entire wire (net). If `top` mode is specified, the wire load model is selected using the area lookup table of the top-level module, which is the default.
- The `set_wire_load_selection_table` selects wire load models based on the wire load model selection table `WireAreaCon` which is defined in the `tpz973gwc_4.3.tlf` library.

Checking Constraints

Cadence recommends that you check all constraints before you execute the `do_optimize` command or any timing analysis commands, such as `report_timing`. This helps you verify that the timing environment is complete and self-consistent.

1. Check the constraints by entering the following command:

```
bg_shell>check_timing
```

The `check_timing` command performs a variety of consistency and completeness checks on the timing constraints specified for a design. Running this command can save you a lot of wasted time. This command can be used on generic and mapped netlists.

The `check_timing` command displays the report shown in Figure 2-5.

Figure 2-5 check_timing Report

Warning	Warning Description	Number of Warnings
clock_but_data	Clock signal found where data is expected	1
clock_clipping_freq	Clock clipping possible due to incompatible clock signal & data signal frequencies	11
data_gating_clock	Data gating clock	6
multiple_signal	Multiple signals arriving at end point	67
no_drive	No drive assertion	1
no_ext	No external delay or required time assertion	2

1. Use the `-verbose` option with the `check_timing` command to look at the details of the warnings:

```
bg_shell>check_timing -verbose
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The `-verbose` option displays a detailed report in addition to the report shown in [Figure 2-5](#) on page 39. An extract of the detailed report is shown in [Figure 2-6](#) on page 40.

The `TIMING CHECK SUMMARY` shown in [Figure 2-5](#) on page 39 includes a summary of warnings, a brief description, and the number of warnings.

The `TIMING CHECK DETAIL` shown in [Figure 2-6](#) on page 40 gives the hierarchical pin names and a description of the warnings.

Figure 2-6 Extract of the Detailed Warning Information

TIMING CHECK DETAIL	
Pin	Warning
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I1	Clock signal (vclk1_int1 neg) found where data is expected
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I1	Multiple signals (vclk1_int1 pos vclk1_int1 neg) arriving
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0	Clock clipping possible due to incompatible clock signal 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I0' and data signals 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/S' frequencies
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0	Clock clipping possible due to incompatible clock signal 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I1' and data signals 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/S' frequencies
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0	Clock clipping possible due to incompatible clock signal 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I1' and data signals 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/S' frequencies
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0	Data gating clock
DTMF_INST/RAM_256x16_TEST_INST/RAM_256x16_INST/A[7]	Multiple signals (vclk1_int1 lead vclk1 lead) arriving
DTMF_INST/RESULTS_CONV_INST/clear_flag_reg/D	Multiple signals (vclk1 lead vclk2 trail) arriving
DTMF_INST/RESULTS_CONV_INST/gt_reg/D	Multiple signals (vclk1 lead vclk2 trail) arriving
....	...



Tip

You can use the `set_table_style` command to change the column widths of the table. For example, to change the column widths of the `TIMING CHECK DETAIL` table use:

```
set_table_style -name report_timing_prologue -max_widths 40
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

All the warnings should be checked, and each warning should be justified or fixed by adding new constraints or modifying existing constraints.

- The following warning indicates that a clock signal is being gated (MUX), where the expected input signal on the combinational gate is data, rather than a clock:

```
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I1 ---- Clock signal (vclk1_int1 neg )  
found where data is expected
```

Since clock gating logic is present in the design, you can ignore this warning.

- The following warning indicates that the data signal at pin I1 of the instance DTMF_INST/DATA_SAMPLE_MUX_INST/i_5 is triggered by the positive edge and negative edge clocks of vclk1_int1:

```
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I1 ---- Multiple signals (vclk1_int1 pos  
vclk1_int1 neg ) arriving
```

- The following message indicates that the logic value of the clock input before the active clock transition may not be controlling the DTMF_INST/DATA_SAMPLE_MUX_INST/i_5 gate, resulting in clock clipping:

```
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0 ---- Clock clipping possible due to  
incompatible clock signal 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/I0' and data  
signals 'DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/S' frequencies
```

If the clock input is not dominant or controlling before and after it makes its transitions, then the output signal could be distorted by the data input. This distortion is called clipping or shaping.

- The following warning message indicates that a single data signal at pin D of instance DTMF_INST/RESULTS_CONV_INST/high_mag_reg_0/D is triggered by the vclk1 and vclk2 clocks:

```
DTMF_INST/RESULTS_CONV_INST/high_mag_reg_0/D ---- Multiple signals (vclk1 lead  
vclk2 trail ) arriving
```

- The following warning indicates that the clock signal at DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0 is being gated:

```
DTMF_INST/DATA_SAMPLE_MUX_INST/i_5/O0 --- Data gating clock
```

Important

As a general strategy, you should either fix or understand and dismiss all `check_timing` errors before moving on to optimization.

Note: `check_timing -old` will give you <TA-XXX> warnings.

Setting Path Groups

You can use path groups to group paths for the purpose of timing optimization or timing analysis. When you have overconstrained paths or when I/Os are overconstrained because of uncertainties, each group can be optimized separately. This enables the tool to work on other paths that have more realistic constraints when attempts to find improvements on the overconstrained paths fail. You can also group the timing paths which have some violations, to do incremental timing optimization.

- Set the path groups for this design, by running the following script:

```
bg_shell> source $tcl_dir/path_groups.tcl
```

The `path_groups.tcl` script contains the following commands

```
set_path_group -name IN -from [ find -input -noclocks * ]
set_path_group_options IN -target_slack 0.2 -all_end_points
```

```
set_path_group -name OUT -to [ find -output * ]
set_path_group_options OUT -target_slack 0.5 -all_end_points
```

```
set_path_group -name vclk2_group -clock_from vclk2
set_path_group_options vclk2_group -target_slack 0.0 -all_end_points
```

```
set_path_group -name vclk1_group -clock_from vclk1
set_path_group_options vclk1_group -target_slack -1.0 -all_end_points
```

The `set_path_group` command groups the critical paths in the design to be optimized. You can use this command in conjunction with the `set_path_group_options` command to potentially specify a different set of optimizations for the grouped paths. All grouped and ungrouped paths are optimized according to the options specified in the `do_optimize` command, unless the grouped paths have different optimization options specified using the `set_path_group_options` command.

In this design you use the `set_path_group` command to create four path groups: `IN`, `OUT`, `vclk2_group`, and `vclk1_group`. Each group has its own optimization requirements which are specified with the `set_path_group_options` command. The target slack for the `vclk1_group` group is set to `-1.0` to show some negative slack after optimization.

Initially, there is only the `default` group. You cannot create any other groups with that name. Each group created with `set_path_group` is separated from the `default` group.

Optimizing the Design

Logic optimization plays a key role in the synthesis process and consists of several processes, including: Boolean transformations, flattening, structuring, technology-independent and technology-dependent mapping, hierarchical optimization, and context derivation. Depending on the design size, the two most commonly used optimization techniques are top-down and bottom-up optimization.

- ▶ Optimize the DTMF design using the top-down synthesis methodology by entering the following command:

```
bg_shell>do_optimize
```

Note: Depending on the performance of your machine, the optimization runs for about 30 minutes. You can take this time to read more about the difference between top-down and bottom-up optimization in [More about Optimizing the Design](#) on page 154.

do_optimize Info Messages

The `do_optimize` command displays several informational messages. Examples of these messages are explained below.

- Message 605 indicates that AmbitWare components (ACL and AWL cells) created as hierarchical instances during the `do_build_generic` step are dissolved in the parent module.

```
Info: Dissolving AmbitWare instance 'i_5377' (cellref 'AWMUX_2_2') in module 'arb' ... <TCLNL-605>.
```

- The following message indicates that the optimizer is creating a unique instance `i_3282` for the multiple instantiated module `AWACL_SUB_UN_162`. This allows the optimizer to optimize one instance module differently from another by placing different constraints on them. The transform to uniquely instantiate multiple instances is `do_uniquely_instantiate`.

```
Info: Duplicated module 'AWMUX_8_3' as 'AWMUX_8_3_1' and bound to instance 'i_3094' in module 'spi' <FNP-700>.
```

- The following message indicates that the optimizer is applying Boolean and algebraic algorithms and transformations to achieve logic optimization and logic structuring. You will get this message for all the modules in the DTMF design. The transform to perform structuring is `do_xform_structure`.

```
Info: Structuring module 'spi' ... <TCLNL-500>.
```

- The following message indicates that the optimizer is propagating logic levels (0 or 1) throughout the design crossing hierarchical boundaries. The transform to perform propagation of constants is `do_xform_propagate_constants`.

```
Info: Propagating constants ... <TCLNL-505>.
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- The following message indicates that the optimizer is removing redundancies from the netlist. The transform to remove redundancy is `do_xform_remove_redundancy`.

Info: Removing redundancies ... <TCLNL-504>.

- The following message indicates that the optimizer is mapping a generic netlist for the `spi` module to the target technology library. You will get similar messages for all the modules in the DTMF design. The transform to perform mapping is `do_xform_map`.

Info: Mapping module 'spi' ... <TCLNL-501>.

- The following report is the Path Group Options report generated during the optimization process. It shows five path groups with their respective effort level, target slack, critical endpoints and worst slack. Incremental reports will be generated as the optimization process continues.

Path Group Options Report						
PathGroup	Effort	All Pts	Target Slack	Critical Endpoints	Worst Slack	TEFS
default	medium	-	0.00000	0/5	9.55629	0.00000
IN	medium	+	0.20000	0/23	2.63263	0.00000
OUT	medium	+	0.50000	16/25	-2.40688	27.60275
vclk1_group	medium	+	-1.00000	326/1124	-5.40008	856.17926
vclk2_group	medium	+	0.00000	16/378	-1.37748	22.03970

Stopping and Restarting Optimization

BuildGates Synthesis provides several features that allow you to stop and restart optimization.

During optimization, you can always issue the `control-c` command to stop the optimization run.

Use the `-checkpoint` option with the `do_optimize` command to periodically save the entire database (`checkpoint.adb`). In the event of a network crash, you can resume `bg_shell` by loading the `checkpoint.db` file.

Use the command, `dump_adb bg_shell_process_id`, with the process ID of a `bg_shell` process, to dump the `.adb` (Ambit Synthesis Database) file at the next most suitable time that will allow the process to be restarted.

Generating Reports

At this point, the synthesis process has transformed your design from RTL to a final technology-mapped netlist that meets all the constraints. Now, you need to capture the information and analyze the reports to see if the synthesis goals are met. You need to run various timing reports to determine:

- Whether the design has any problems
- Whether the design passes or fails timing analysis and where the violations are
- The constraints and exceptions for timing analysis
- How the delays are calculated

BuildGates Synthesis provides a number of report categories such as timing, area, hierarchy, library, design rule violations, end point slack and path histograms, fanin and fanout, finite state machine to analyze various steps in synthesis process.

In this module, you will only use a subset of all the possible report options.

Note: One of the key advantages of BuildGates Synthesis is that it has a built-in, full-chip, sign-off quality, static timing analysis capability. You do not need to use a separate standalone STA tool to perform timing analysis. This makes BuildGates more efficient and accurate and eliminates the need to buy another tool to signoff your design.

Generating Summary and Detailed Reports

1. Generate a summary report for all timing checks.

```
bg_shell>report_analysis_coverage > $rep_dir/report_analysis_coverage.rpt
```

[Figure 2-7](#) on page 46 shows an excerpt of the generated report.

- TIMING CHECK COVERAGE SUMMARY has a summary of all timing checks. For each timing check, the report indicates whether the timing condition was met, violated, or untested.
- TIMING CHECK COVERAGE DETAILS gives details of each pin and reference pin. The report shows that untested setup checks are due to “No data signal” on the scan pins. You can ignore these checks, since scan pins are in tieback mode at this stage.

The Pin and Reference Pin columns represent hierarchical pins `DTMF_INST/ARB_INST/dma_grant_reg/*`, and `DTMF_INST/ARB_INST/dma_grant_reg/*`, respectively. The `report_analysis_coverage.rpt` file has the complete pin names.

Cadence Synthesis Rapid Adoption Kit Getting Started with BuildGates Synthesis

Figure 2-7 Excerpt of the Summary Report

Report	report_analysis_coverage			
Options	> ../rpt/report_analysis_coverage.rpt			
Date	20030522.115114			
Tool	bg_shell			
Release	v5.10-s058			
Version	May 20 2003 13:14:56			

TIMING CHECK COVERAGE SUMMARY				
Check Type	No. of Checks	Met	Violated	Untested
ClockGatingHold	24	0 (0%)	24 (100%)	0 (0%)
ClockGatingSetup	24	24 (100%)	0 (0%)	0 (0%)
ClockPeriod	6	6 (100%)	0 (0%)	0 (0%)
ExternalDelay (Early)	50	50 (100%)	0 (0%)	0 (0%)
ExternalDelay (Late)	50	50 (100%)	0 (0%)	0 (0%)
Hold	3632	1442 (39%)	18 (0%)	2172 (59%)
PulseWidth	1618	1092 (67%)	0 (0%)	526 (32%)
Recovery	269	6 (2%)	0 (0%)	263 (97%)
Setup	3632	1318 (36%)	142 (3%)	2172 (59%)

TIMING CHECK COVERAGE DETAILS				
Pin	Reference Pin	Check Type	Slack	Reason
dma_grant_reg/CK ^	dma_grant_reg/CK v	PulseWidth	2.36	
dma_grant_reg/CK v	dma_grant_reg/CK ^	PulseWidth	2.11	
dma_grant_reg/D ^	dma_grant_reg/CK ^	Setup	0.35	
dma_grant_reg/D ^	dma_grant_reg/CK ^	Hold	1.23	
dma_grant_reg/D v	dma_grant_reg/CK ^	Setup	-0.00	
dma_grant_reg/D v	dma_grant_reg/CK ^	Hold	1.38	
dma_grant_reg/RN	dma_grant_reg/RN	PulseWidth	UNTESTED	No reference signal
dma_grant_reg/RN ^	dma_grant_reg/CK ^	Recovery	UNTESTED	No data signal
dma_grant_reg/SE	dma_grant_reg/CK ^	Hold	UNTESTED	No data signal
dma_grant_reg/SE	dma_grant_reg/CK ^	Setup	UNTESTED	No data signal
dma_grant_reg/SI	dma_grant_reg/CK ^	Setup	UNTESTED	No data signal
...

The `report_analysis_coverage` command allows you to generate detailed reports for any type of checks shown above.

Cadence Synthesis Rapid Adoption Kit Getting Started with BuildGates Synthesis

2. Generate a detailed report on the setup timing check:

```
bg_shell>report_analysis_coverage -check_type setup -sort slack \  
> $rep_dir/setup.rpt
```

An excerpt of the detailed report on the setup timing check is shown in Figure 2-8.

Figure 2-8 Summary of Setup Timing Check

Report	report_analysis_coverage
Options	-check_type setup -sort slack > ../rpt/setup.rpt
Date	20030522.123427
Tool	bg_shell
Release	v5.10-s058
Version	May 20 2003 13:14:56

TIMING CHECK COVERAGE SUMMARY				
Check Type	No. of Checks	Met	Violated	Untested
Setup	3632	1318 (36%)	142 (3%)	2172 (59%)

TIMING CHECK COVERAGE DETAILS				
Pin	Reference Pin	Check Type	Slack	Reason
...	...			
r1200_reg_10/D v	r1200_reg_10/CKN v	Setup	-0.02	
r1477_reg_10/D v	r1477_reg_10/CKN v	Setup	-0.02	
r697_reg_10/D v	r697_reg_10/CKN v	Setup	-0.02	
r770_reg_1/D v	r770_reg_1/CKN v	Setup	-0.02	
r770_reg_11/D ^	r770_reg_11/CKN v	Setup	-0.02	
r770_reg_9/D v	r770_reg_9/CKN v	Setup	-0.02	
...	

Note: The Pin and Reference Pin columns represent hierarchical pins DTMF_INST/RESULTS_CONV_INST.

The last table shows that the report sorted all the setup checks in increasing order by slack. At this time, the design has some setup violations.

Checking Setup Violations

- Generate a timing report to find out if the design works at the specified frequency or to see the worst timing path in the design:

```
bg_shell>report_timing > $rep_dir/setup_timing.rpt
```

The timing report contains the start and end nodes of the path and the delay through the entire path. The `report_timing` command has a number of options, such as `-from`, `-through`, and `-to`, to specify the begin, through, and end points (respectively) of the path, `-max_points`, and `max_slacks nworst`.

Figure 2-9 on page 49 shows a typical timing report:.

- The first section lists the options used with `report_timing`, tool version, process, voltage, temperature, operating conditions, top timing module (DTMF_chip), and so on.
- The second section describes the beginning and ending points of the path, launching and capturing edge of the clock, whether timing on the path is met or violated. For detailed cell-to-cell delays, see the `setup_timing.rpt` file.

Note: As mentioned before, your results might differ from what is shown here due to difference in runs and release versions.

For setup check calculation, the tool uses the worst case delays of the library cells. By default, `report_timing` without any options checks the setup using worst case delays.

- The third section of the timing report is a table describing all the instances on the path—timing arc, cell name, delay through timing-arc, the signal required times, and the actual signal arrival times at the output of each instance.

Cadence Synthesis Rapid Adoption Kit Getting Started with BuildGates Synthesis

Figure 2-9 Excerpt of the Setup Time Report

```

+-----+-----+
| Report           | report_timing |
+-----+-----+
| Options          | > ../rpt/setup_timing.rpt |
+-----+-----+
| Date            | 20030522.125015 |
| Tool            | bg_shell       |
| Release         | v5.10-s058    |
| Version         | May 20 2003 13:14:56 |
+-----+-----+
| Module          | dtmf_chip     |
| Timing          | LATE          |
| Slew Propagation | WORST        |
| Operating Condition | slow       |
| PVT Mode        | max           |
| Tree Type       | balanced      |
| Process         | 1.00000      |
| Voltage         | 1.62000      |
| Temperature     | 125.00000    |
| time unit       | 1.00000 ns   |
| capacitance unit | 1.00000 pF   |
| resistance unit  | 1.00000 kOhm |
+-----+-----+

```

```

Path 1: VIOLATED Setup Check with Pin DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST
data_out_reg_0/CK
Endpoint: DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/data_out_reg_0/D (v)
checked with leading edge of 'vclk1'
Beginpoint: DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/arp_reg/Q (^)
triggered by leading edge of 'vclk1'
Path Groups: {vclk1_group}
Other End Arrival Time 0.00000
- Setup 0.69730
+ Phase Shift 6.00000
- Uncertainty 0.25000
= Required Time 5.05270
- Arrival Time 5.75437
= Slack Time -0.70167
  Clock Rise Edge 0.00000
  = Beginpoint Arrival Time 0.00000

```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	refclk ^			0.00000	-0.70167
IOPADS_INST	refclk ^	iopads		0.00000	-0.70167
IOPADS_INST/Prefclkkip	PAD ^ -> C ^	PDIDGZ	0.00000	0.00000	-0.70167
IOPADS_INST	refclkI ^	iopads		0.00000	-0.70167
...

You can use the `-format` option to customize the reports to your needs by requesting the exact fields in which you have an interest. Using a combination of the `-format` and

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

`-tcl_list` options lets you integrate the timing reports into your Tcl scripts. You can write the report to a file by specifying the filename. You can also set the global variable `report_timing_format` to specify the format for the `report_timing` command.

BuildGates Synthesis comes with a host of sign-off quality, full-chip, static-timing-analysis features such as:

- Clock reconvergence pessimism removal
- Pulse width and recovery and removal checks
- Full path and end point histograms
- Simultaneous bc/wc analysis
- Full support for all major back annotation formats (`sdf`, `dspf`, `spef`, `rspf`)
- Path group capability
- Support for common exceptions such as `false_path`, `multicycle_path`, and so on
- STAMP modeling and so on.

See the [*Common Timing Engine \(CTE\) User Guide*](#) for more information.

Generating Area Reports

- Generate an area report.

```
bg_shell>report_area > $rep_dir/area.rpt
```

The area report will look similar to:

```
+-----+
| Report | report_area |
+-----+
| Options | > ../rpt/area.rpt |
+-----+
| Date   | 20030522.125511 |
| Tool   | bg_shell        |
| Release | v5.10-s058      |
| Version | May 20 2003 13:14:56 |
+-----+
| Module | dtmf_chip       |
+-----+
```

```
Summary Area Report
Source of Area : Timing Library
-----
```

Cadence Synthesis Rapid Adoption Kit Getting Started with BuildGates Synthesis

Module	Wireload	Cell Area	Net Area	Total Area
dtmf_chip	TSMC18_Conservative	1392217.23010	0.00	1392217.23010
dtmf_recvr_core	TSMC64K_Conservative	579117.23010	0.00	579117.23010
iopads	TSMC64K_Conservative	813100.00000	0.00	813100.00000
arb	TSMC8K_Conservative	671.93282	0.00	671.93282
data_sample_mux	TSMC8K_Conservative	678.58561	0.00	678.58561
digit_reg	TSMC8K_Conservative	725.15523	0.00	725.15523
dma	TSMC8K_Conservative	2022.45128	0.00	2022.45128
ram_128x16_test	TSMC8K_Conservative	100791.69418	0.00	100791.69418
ram_256x16_test	TSMC16K_Conservative	113656.55447	0.00	113656.55447
results_conv	TSMC16K_Conservative	42464.82314	0.00	42464.82314
spi	TSMC8K_Conservative	2461.53600	0.00	2461.53600
...

The table shows that the total area of the top module (dtmf_chip) is 1.393.684 square microns, as shown in the first row of the table.

The area report table contains all the modules in the design—the wireload model name based on wire load section table, the total cell area, the net area and the total area for each module. You can use the `-cells` option to get more details on cell names and counts used in the current module, area for each cell, and total cell area.

Writing Netlist and Database Files

At this stage, optimization is complete and you find the report results satisfactory. Next, you need to save the technology-dependent, optimized netlist and BuildGates Synthesis database file.

1. Save the optimized netlist.

```
bg_shell>write_verilog -hierarchical $adb_dir/dtmf_chip_opt.vs
```

The `write_verilog` command writes out a hierarchical, mapped netlist (dtmf_chip.v) that is stored in the database in Verilog format. If you write a netlist after the `do_build_generic` command, it saves a generic-level netlist which contains instances of ATL and XATL cells.

Note: You can use the `write_vhdl` command to write out a netlist in VHDL format and the `write_edif` command to write out a netlist in edif format. If you get assign statements in the Verilog netlist, you can use the transform, `do_xform_fix_multiport_nets`, or the global variable, `set_global_fix_multiport_nets`, to avoid assign statements.

2. Save the database file.

```
bg_shell>write_adb $adb_dir/dtmf_chip_opt.adb
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The `write_adb` command writes the design data for the `dtmf_chip` design in BuildGates Synthesis database (ADB) file format. By default, the ADB netlist is a hierarchical netlist of the current module and all instances inside it.

You can use the `read_adb` command to quickly load design data in `bg_shell` for further analysis or synthesis. The `write_adb` command has several useful options such as the `-no_assertions` option to write out the database without timing constraints, the `-no_rc` option to write out the database without any wire resistances or capacitances, and so on.

Setting Scan Chain Assertions

So far, you ran test-synthesis in `tieback` mode. In this mode, the test synthesis tool connects the scan-data output pin of the scan register to its own scan-data input pin. The `tieback` mode speeds up synthesis due to the fact that chain path connections are not being considered. Hence, Cadence recommends that you perform your initial synthesis in `tieback` mode using BuildGates Synthesis. Now that optimization is completed, you can set scan chain assertions and connect the scan chain.

To connect the scan chain, you need to set the following assertions:

- The number of scan chains and/or the maximum length of the scan chains
- The compatible clock domains for data lockup latch analysis and/or insertion
- The scan-data input/output pairs

1. Set the scan chain assertions using the following script:

```
bg_shell>source $tcl_dir/connect_scan_chain.tcl
```

The `connect_scan_chain.tcl` script contains the following commands:

```
set_number_of_scan_chains 3
```

```
set_scan_data {IOPADS_INST/Ptdspip00/C} {IOPADS_INST/Ptdspop00/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip01/C} {IOPADS_INST/Ptdspop01/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip02/C} {IOPADS_INST/Ptdspop02/I} -shared_out
```

```
set_dft_compatible_clock_domain -same_clock
set_global dft_scan_path_connect chain
```

```
do_xform_connect_scan
```

- The `set_number_of_scan_chains` command specifies the number of scan chains (3) to create for the current module (DTMF). Use the `set_max_scan_chain_length` assertion to specify a maximum scan chain length. Meeting the specified number of scan chains takes priority over meeting a specified maximum chain length, unless the later assertion is specified with the `-priority` option. By default, the tool creates one chain for each clock domain.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

- ❑ Each scan chain needs its own input and output port. The `set_scan_data` assertion specifies the names for the input (`IOPADS_INST/ptdspip0*/I`) and output (`IOPADS_INST/ptdspip0*/C`) scan data ports. In addition, the `-clock` option associates the input and output names with a specific DFT clock domain. The `-shared_out` option enables the scan data output port to be shared with a functional port using a MUX.
- ❑ By default, the test synthesis tool creates one scan chain for each DFT clock domain. The `set_dft_compatible_clock_domain` assertion specifies domain and phase compatibility between the DFT clock domains. You must specify this command prior to running the scan connection engine in chain mode. It instructs the scan connection engine to create a scan chain in which a data lockup element (latch or flip-flop) can be optionally inserted between the scan chain segments belonging to the different DFT clock domains. The `-same_clock` option specifies that both edges of each clock are compatible.
- ❑ The `set_global dft_scan_path_connect` assertion sets the connection mode. Before optimization, you set this global variable to `tieback` mode.) In `chain` mode, the test synthesis tool connects the scan-data output pin from one register to the scan-data input pin of the next register, thus creating the top-level scan chains.
- ❑ The `do_xform_connect_scan` command configures and connects scan flip-flops into scan chains. The command applies to the current module and all lower hierarchies referred to in the current module. The design must be mapped before connecting scan chains in a design. All flip-flops that pass DFT rule check (through the `check_dft_rules` command) are converted to scan flip-flops and are connected into scan chains. The flip-flops that do not pass DFT rule checks are automatically excluded from scan chains. The flip-flops in the lower module hierarchies are also automatically reconfigured and connected, unless the lower module is tagged with `dont_modify` or `dont_touch_scan`.

2. Examine the messages you get while running this script.

- ❑ By specifying the `set_dft_compatible_clock_domains -same_clock` assertion, domains can be merged for this design. When the top-level scan chains are created, the negative edge-triggered flops, belonging the DFT clock domain 0, will proceed the positive edge-triggered flops, belonging to DFT clock domain 1. Test synthesis will not insert a data lockup element between the domain transitions of the chain segments triggered by the alternative phases of `scan_clk` when constructing the top level chain.

For scan flops belonging to different root clock domains, if domain merging was specified, test synthesis would insert a data lockup element between the domain transitions of the scan flops belonging to the different DFT domains or between gated-clock branches of the same top-level logical clock source, `refclk`. In this situation, the following Info message would be reported:

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Info: Inserting lockup latch (f/f) instance '<path_to_lockup_element>' to connect scan flip-flops '<path_to_flip_flop>' and '<path_to_flip_flop>' in a single chain <DFT-344>.

Insertion of data lockup elements is only supported for the multiplexed scan (muxscan) style of test insertion. Insertion of data lockup elements allows you to reduce the number of scan chains configured in a multi-clock domain design. Additionally, insertion of data lockup elements between different gated-clock branches, derived from the same logical clock source, can prevent capture problems during the scan-shift cycle of the test mode due to clock skew variances between the different gated-clock branches. Without this capability, test synthesis creates a single scan chain for each DFT clock domain.

- ❑ The following message indicates that the test synthesis tool created a flat scan report file called `dtmf_chip.scan.flat`.

```
DFT - creating flat scan order file 'dtmf_chip.scan.flat'
```

The flat scan chain report file is organized by scan chain, listing all the bits on each scan chain in the order of connection.

Note: You can also create a hierarchical scan report file using the `write_scan_order_file` command with the `-hier` option.

```
DFT - creating hierarchical scan order file 'dtmf_chip.scan'
```

The hierarchical scan chain report file is organized by module, listing the chain connections visible at each level of hierarchy.

At this stage the scan chain is hooked-up.

3. Report the timing to check the impact of the scan insertion.

```
bg_shell>report_timing -late > $rep_dir/setup_scan_timing.rpt
```

You will notice that, after hooking up scan, the timing becomes worse, which is due to the scan chain order. By default, the test synthesis tool orders the scan chain alphabetically, which is not optimal.

```
Path 1: VIOLATED Setup Check with Pin DTMF_INST/TDSP_CORE_INST/
DATA_BUS_MACH_INST/data_out_reg_0/CK
Endpoint: DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/data_out_reg_0/D (v)
checked with leading edge of 'vclk1'
Beginpoint: DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/arp_reg/Q (^)
triggered by leading edge of 'vclk1'
Path Groups: {vclk1_group}
Other End Arrival Time 0.00000
- Setup 0.69730
+ Phase Shift 6.00000
- Uncertainty 0.25000
= Required Time 5.05270
- Arrival Time 5.80546
= Slack Time -0.75276
Clock Rise Edge 0.00000
= Beginpoint Arrival Time 0.00000
```


Incremental Timing Optimization

Before doing an incremental timing optimization to fix the negative slack, change the target slack of the path group, `vclk1_group` to 0.0 from -1.0 set earlier.

1. Change the target slack.

```
bg_shell>set_path_group_options vclk1_group -target_slack 0.0 -all_end_points
```

2. Do an incremental optimization of the slack.

```
bg_shell> do_optimize -incremental -dont_reclaim_area
```

The `-incremental` option indicates that the design is already well-optimized and that the design rule violations have been fixed. The optimizer works incrementally on the paths which have timing violations.

The `-dont_reclaim_area` option prevents downsizing and the removal of buffers or clone instances to reduce area. By default, area reclamation is done when slack fixing ceases to find improvement. This option is useful when you want prevent area reclamation in parts of the design that are not timing critical.

For more information on scan, see [*Design for Test \(DFT\) Using BuildGates Synthesis and Cadence PKS.*](#)

3. Check the timing again.

```
bg_shell> report_timing -late -nworst 5 > \
$rep_dir/setup_scan_incr_opt_timing.rpt
```

This command generates five worst timing paths between the flip flops in the design.

The report is shown in [Figure 2-10](#) on page 56.

As you can see, the timing constraints are now met.

4. Find out if there are any hold violations in the design.

```
bg_shell> report_timing -early > $rep_dir/time_hold.rpt
```

For hold time calculation you specify the `-early` option to ensure that the tool uses the best case delays of the library cells.

[Figure 2-11](#) on page 57 shows a typical timing report.

Cadence Synthesis Rapid Adoption Kit Getting Started with BuildGates Synthesis

Figure 2-10 Timing Report after Reoptimization

```

+-----+-----+
| Report           | report_timing |
+-----+-----+
| Options          | -late -nworst 5 > ../rpt/setup_scan_incr_opt_timing.rpt |
+-----+-----+
| Date            | 20030522.132637 |
| Tool            | bg_shell       |
| Release         | v5.10-s058    |
| Version         | May 20 2003 13:14:56 |
+-----+-----+
| Module          | dtmf_chip      |
| Timing          | LATE           |
| Slew Propagation | WORST         |
| Operating Condition | slow         |
| PVT Mode        | max            |
| Tree Type       | balanced       |
| Process         | 1.00000        |
| Voltage         | 1.62000        |
| Temperature     | 125.00000     |
| time unit       | 1.00000 ns    |
| capacitance unit | 1.00000 pF    |
| resistance unit  | 1.00000 kOhm  |
+-----+-----+

```

```

Path 1: MET Setup Check with Pin DTMF_INST/TDSP_CORE_INST/EXECUTE_INST
sel_op_a_reg_2/CK
Endpoint:  DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/sel_op_a_reg_2/E (^) checked
with leading edge of 'vclk1'
Beginpoint: DTMF_INST/TDSP_CORE_INST/DECODE_INST/ir_reg_15/Q      (v) triggered
by leading edge of 'vclk1'
Path Groups: {vclk1_group}
Other End Arrival Time  0.00000
- Setup                 1.03155
+ Phase Shift           6.00000
- Uncertainty           0.25000
= Required Time         4.71845
- Arrival Time          4.71031
= Slack Time            0.00814
  Clock Rise Edge      0.00000
  = Beginpoint Arrival Time 0.00000

```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
IOPADS_INST	refclk ^			0.00000	0.00814
IOPADS_INST/Prefclkip	refclk ^	iopads		0.00000	0.00814
IOPADS_INST	PAD ^ -> C ^	PDIDGZ	0.00000	0.00000	0.00814
IOPADS_INST	refclkI ^	iopads		0.00000	0.00814
DTMF_INST	refclk ^	dtmf_recvr_core		0.00000	0.00814
DTMF_INST/PLLCLK_INST	refclk ^ -> clk2x ^	pllclk	0.00000	0.00000	0.00814
...

Cadence Synthesis Rapid Adoption Kit Getting Started with BuildGates Synthesis

Figure 2-11 Excerpt of the Hold Time Report

Report	report_timing
Options	-early > ../rpt/hold_timing.rpt
Date	20030522.133440
Tool	bg_shell
Release	v5.10-s058
Version	May 20 2003 13:14:56
Module	dtmf_chip
Timing	EARLY
Slew Propagation	WORST
Operating Condition	slow
PVT Mode	max
Tree Type	balanced
Process	1.00000
Voltage	1.62000
Temperature	125.00000
time unit	1.00000 ns
capacitance unit	1.00000 pF
resistance unit	1.00000 kOhm

Path 1: VIOLATED Clock Gating Hold Check with Pin DTMF_INST/TDSP_DS_CS_INST/i_10/D
 Endpoint: DTMF_INST/TDSP_DS_CS_INST/i_10/AN (v) checked with trailing edge of 'vclk1_int1'
 Beginpoint: DTMF_INST/TDSP_CORE_INST/PORT_BUS_MACH_INST/as_reg/Q (v) triggered by leading edge of 'vclk1'
 Path Groups: {vclk1_group}
 Other End Arrival Time 6.00000
 + Clock Gating Hold 0.00000
 + Phase Shift 0.00000
 + Uncertainty 0.25000
 = Required Time 6.25000
 Arrival Time 0.37220
 Slack Time -5.87780
 Clock Rise Edge 0.00000
 = Beginpoint Arrival Time 0.00000

Instance	Arc	Cell	Delay	Arrival Time	Required Time
IOPADS_INST	refclk ^			0.00000	5.87780
IOPADS_INST/Prefclkip	refclk ^	iopads		0.00000	5.87780
IOPADS_INST	PAD ^ -> C ^	PDIDGZ	0.00000	0.00000	5.87780
IOPADS_INST	refclkI ^	iopads		0.00000	5.87780
DTMF_INST	refclk ^	dtmf_recvr_core		0.00000	5.87780
DTMF_INST/PLLCLK_INST	refclk ^ -> clk2x ^	pllclk	0.00000	0.00000	5.87780
DTMF_INST/ TEST_CONTROL_INST	clk ^	test_control		0.00000	5.87780
DTMF_INST/ TEST_CONTROL_INST/i_198	A ^ -> Y ^	MX2X1	0.00000	0.00000	5.87780
DTMF_INST/ TEST_CONTROL_INST	m_clk ^	test_control		0.00000	5.87780
...

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The report shows that the design has some hold violations between the clock domains vclk1 and vclk2. You can fix these hold violations with the `do_xform_fix_hold` transform.

You can read [More about Timing Reports](#) on page 156.

5. Write out the Verilog netlist.

```
bg_shell> write_verilog -hierarchical $adb_dir/dtmf_chip_scan.vs
```

6. Save the database file to use when you try the graphical interface ([Using BuildGates Synthesis Graphical User Interface](#) on page 61).

```
bg_shell>write_adb $adb_dir/dtmf_chip_scan.adb
```

7. End this BuildGates session.

```
bg_shell>exit
```

Interfacing with Physical Design Tools

So far, you have been optimizing your design using wire-loads provided with the libraries. For deep sub-micron designs, wireloads are a source of timing closure issues. Addressing timing closure issues requires a coordinated effort in synthesis and place-and-route. BuildGates Synthesis provides all the standard interfaces to floorplanning and place and route tools. Supported interfaces are SDF (2.0 or 3.0 format), PDEF (2.0 IEEE version), Cadence General Constraint Format (GCF, version 1.3 or 1.4), and wire resistance and capacitance files (SPEF, RSPF and DSPF).

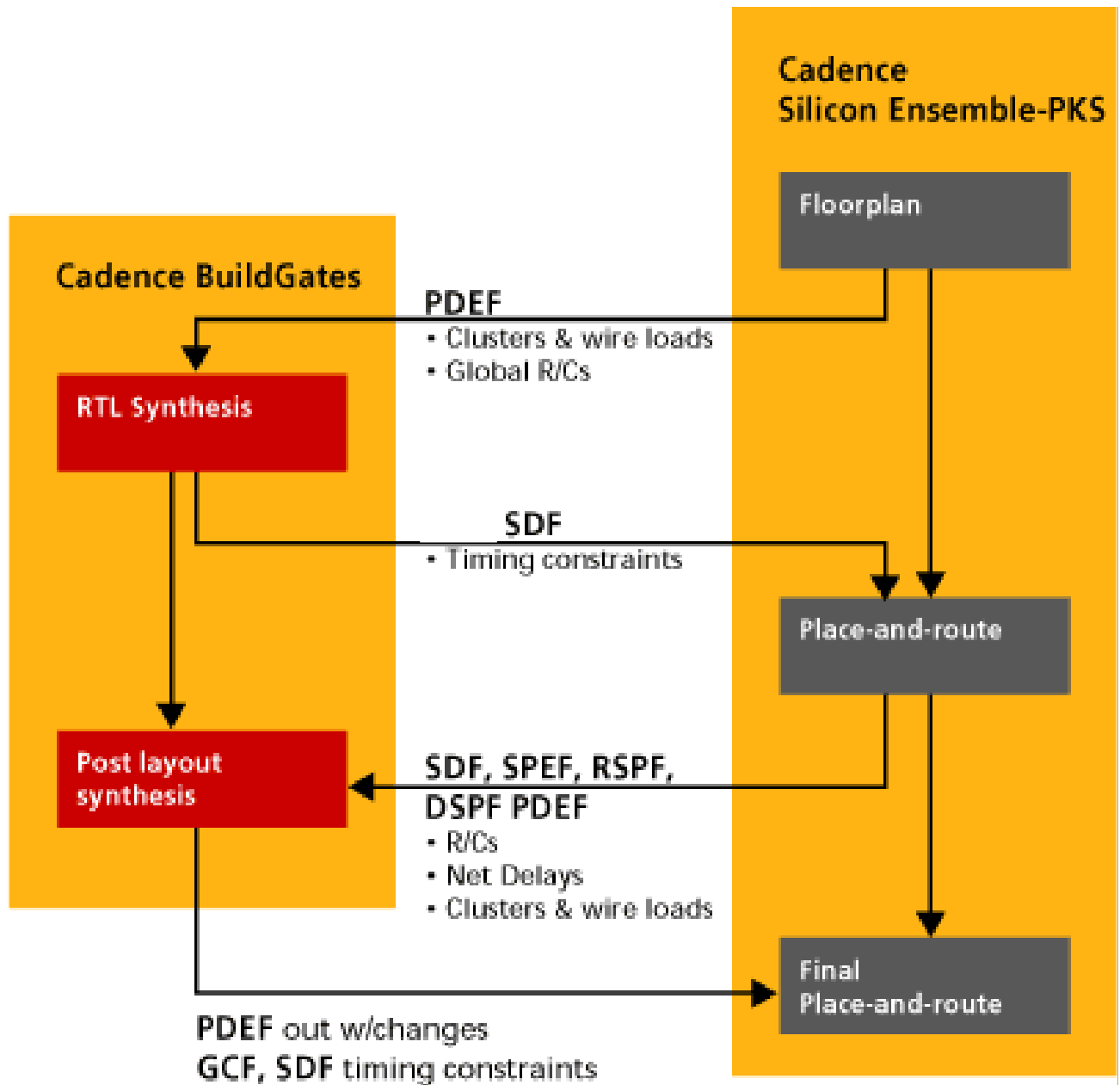
Figure 2-12 on page 60 shows the physical design tool interfaces. Because of the high capacity of BuildGates Synthesis, you can perform in-place and post-layout optimization at chip-level for multi-million gate designs. The transform to perform in-place optimization is `do_xform_ipo`. Important options for this command are `-change_limit` to limit the number of changes and `-change_file` to write out all the changes in a file. To find out more information on interfacing with physical design tools, refer to Optimization With Backannotation in the *BuildGates Synthesis User Guide*.

Figure 2-12 shows that with the traditional timing closure flow, you have to go through several iterations between synthesis and place-route tools. To solve this problem, you can use the Cadence Physically Knowledgeable Synthesis (PKS) tool. It produces perfect timing correlation between the synthesis and place-and-route domains by providing concurrent logical, physical, and timing optimization, thus eliminating iterations between synthesis and place-and-route. PKS provides optimization, placement, global routing, and clock-tree synthesis, all using the same sign-off quality static timing analysis engine.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Figure 2-12 Physical Design Tool Interfaces



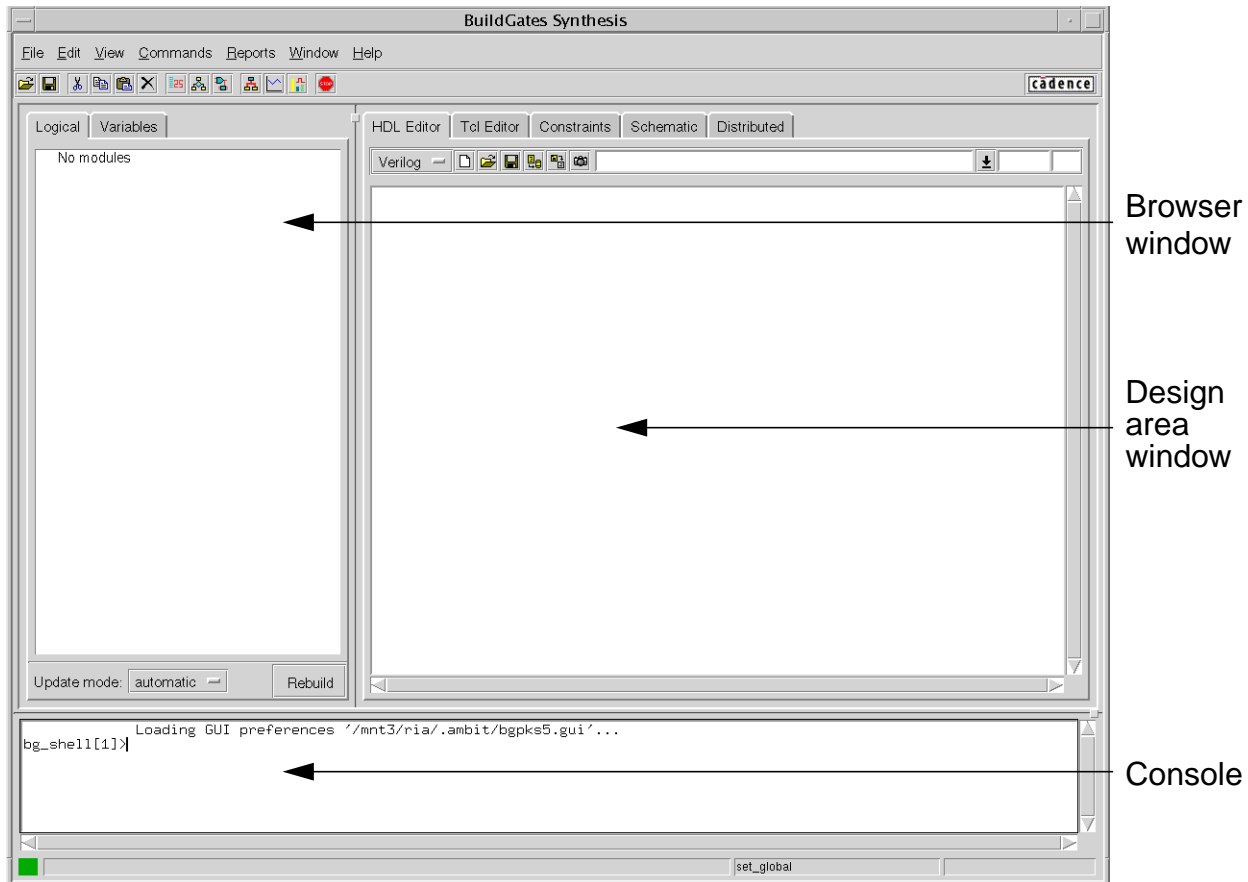
Using BuildGates Synthesis Graphical User Interface

BuildGates Synthesis has an integrated, easy-to-use, graphical user interface (GUI) oriented towards design analysis. Unlike traditional GUIs oriented towards menu-driven tool control, this GUI uniquely combines linked graphical and textual views to speed access to design information, making for more productive design analysis. This GUI also contains constraint specification menus, schematic generation, a slack and area histogram; and linked graphical, schematic, and textual reports.

1. Invoke BuildGates Synthesis in graphical mode.

```
bg_shell -gui &
```

The BuildGates Synthesis window is displayed as shown below:



The GUI has three basic panes.

- The browser window is used to browse the logical hierarchy of your design or variables set in your synthesis run.
- The design area window is primarily used to view schematics and highlighting timing

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

paths. You can also use it to edit Tcl, hdl, constraints, and so on.

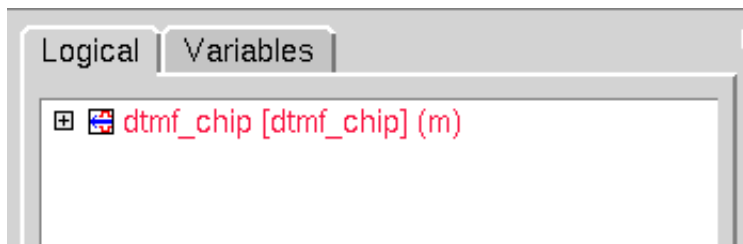
- The console is the command line interface that you have been working with. All commands that you can click on are echoed in this pane to make it easier to learn the tool.

Any pane can be selected by clicking in the pane, and <CTRL-M> can be used to maximize or minimize the selected pane.

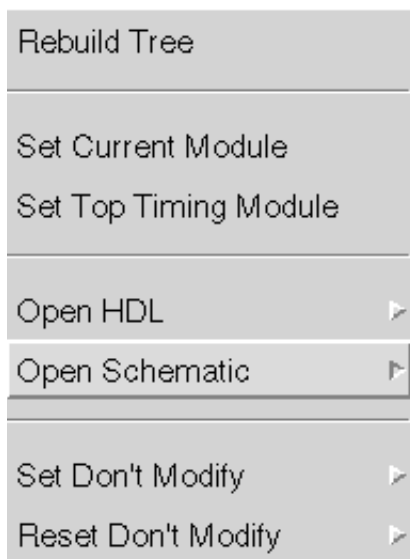
2. Enter the following commands in the console:

```
source ../tcl/setup.tcl
source $tcl_dir/set_globals.tcl
source $tcl_dir/read_lib.tcl
read_adb $adb_dir/dtmf_chip_scan.adb
```

The following appears in your Logical browser:



3. Click left on *dtmf_chip* to select it.
4. Click right in the browser window to bring up the popup menu.

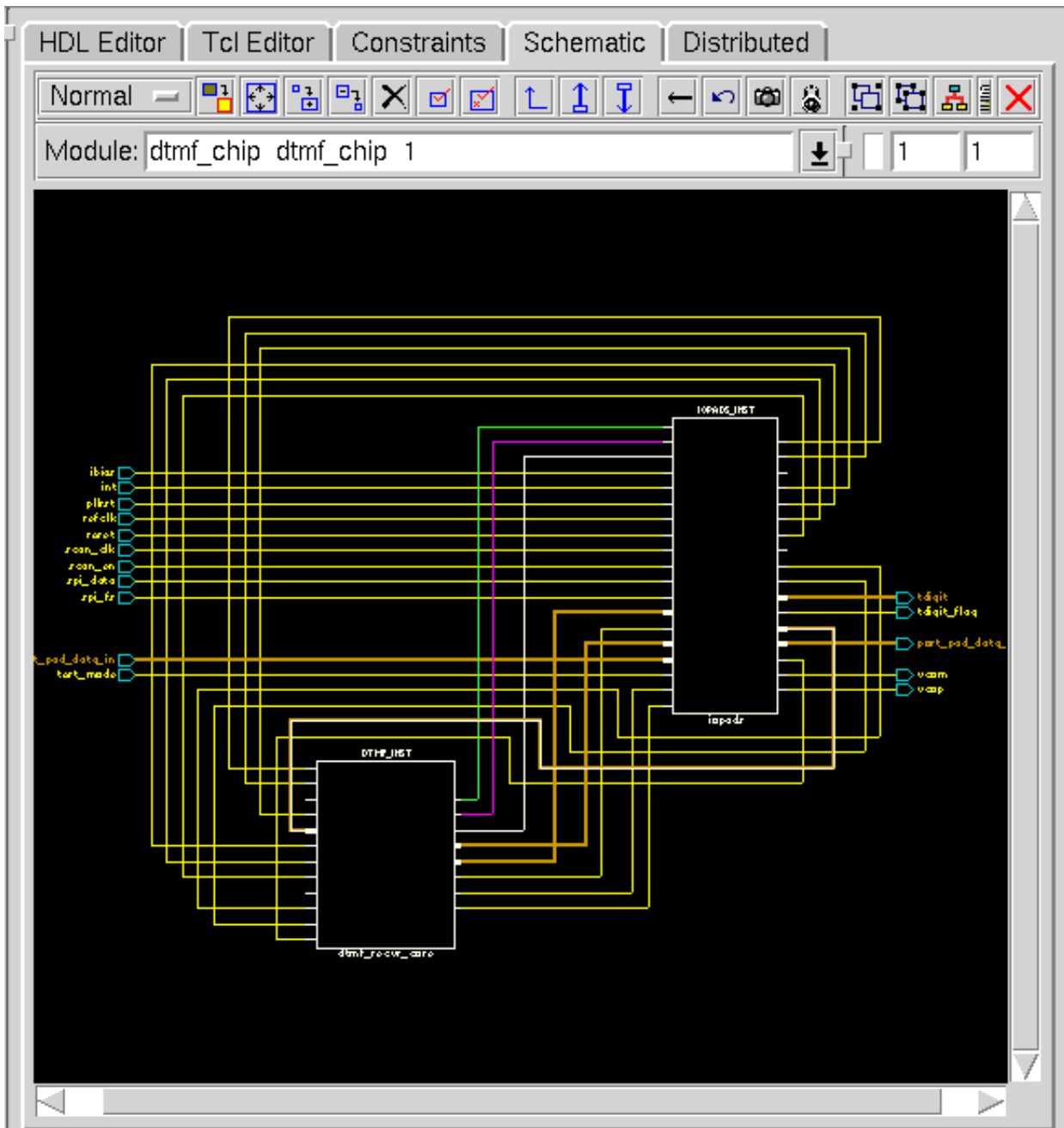


Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

5. Select *Open Schematic – Main Window*.

The schematic of the DTMF chip is displayed in the Schematic window.



The colored paths correspond to the three scan chains that you configured earlier.

With the left mouse you can zoom out, go up in the hierarchy, zoom in, or zoom to fit. A double click will dig deeper in the hierarchy. The middle mouse button is used to pan

Cadence Synthesis Rapid Adoption Kit

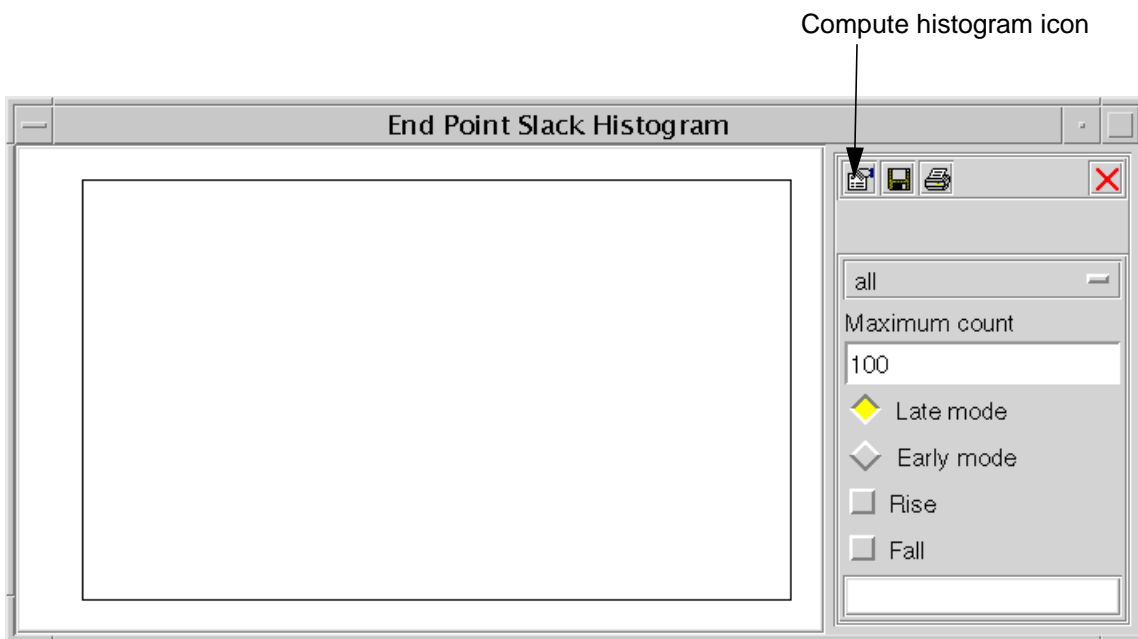
Getting Started with BuildGates Synthesis

around the schematic. The right mouse button is used to first select what is under the mouse (general schematic, net, port, pin, or module), then to provide special functions for that item.

For more information on the GUI, see [GUI Guide for BuildGates Synthesis and Cadence PKS](#)

6. To generate the end point slack histogram for the 100 worst paths, select *Reports – End Point Histogram*. The End Point Slack Histogram window shown in Figure 2-13 appears.

Figure 2-13 End Point Slack Histogram

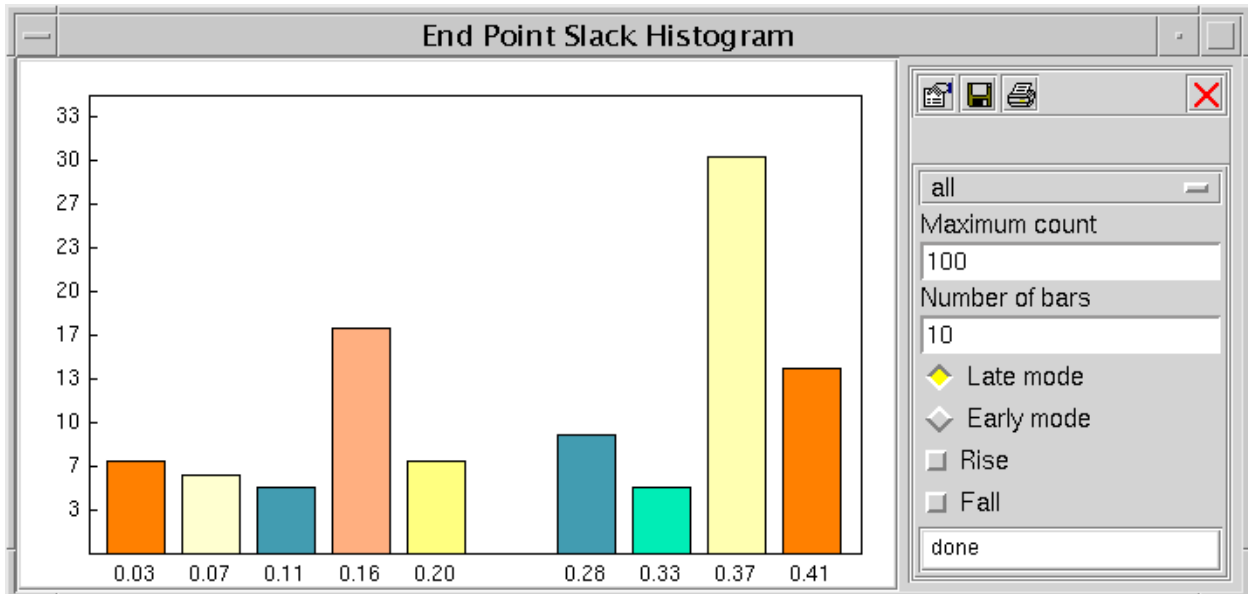


- a. Keep *Late mode* selected.
- b. Select the Compute Histogram icon shown in [Figure 2-13](#) on page 64.

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The generated End Point Slack Histogram shows 100 paths falling in different slack windows. You will see something similar to:



You can also write the results into a file.

c. Click on the Close window icon (red x) to close the histogram window.

1. To generate the End Point histogram for 100 the best/early paths, select *Reports – End Point Histogram* and select *Early mode* on the window. The results can also be written to a file.

Similarly, you can generate Path Histogram Reports. For a complete description of Path histograms, refer to *GUI Guide for BuildGates Synthesis and Cadence PKS.*

2. To end the session, select *File – Exit* or type `exit` in the console.

Migrating to BuildGates from Design Compiler

You are now ready to try out BuildGates Synthesis on your own design following these steps. If you already have Synopsys Design Compiler (DC) or Prime Time (PT) timing constraints, you can use the BuildGates Synthesis Constraint Translator to translate DC or PT output to a BuildGates Synthesis constraints file.

1. Locate your library in `.alf` or `.tlf` format. If you do not have a `.alf` library, ask your vendor or convert a `.lib` to `.alf` or `.tlf`. For more information, see the [Using Timing Libraries](#) chapter of the *Common Timing Engine (CTE) User Guide*.
2. Determine which product you need:
 - a. BuildGates Synthesis is suitable for most synthesis needs
 - b. BuildGates Synthesis Extreme includes everything that BuildGates Synthesis has, as well as advanced engines for Low Power and Datapath optimization. Datapath includes the basic arithmetic functions (+,-,*) as well as the comparison functions (==, <=, and so on).
 - c. PKS is an advanced physically knowledgeable synthesis tool that can be used if you cannot meet timing with traditional tools.
3. Convert your Synopsys Design Compiler (DC) constraints to BuildGates synthesis constraints.
 - a. BuildGates Synthesis only needs top-level constraints, so you can remove most of your lower, block-level I/O constraints. You will need to retain any other lower level constraints, such as `false_path` and `multi_cycle_paths`.
 - b. Use the `write_script` command in DC or PT to create an output file (in `dclsh` or `Tcl` format).
 - c. Use the `read_dc_script` in BuildGates Synthesis to read your constraints.
Note: You can only run the `read_dc_script` command after the `do_build_generic` step.

See [Constraint Translation Example](#) on page 67 for an example.

Most designs can be easily constrained with only a few commands which set up the clocks, I/O constraints, and special path settings.

The following table shows Synopsys DC Compiler commands and the corresponding BuildGates BuildGates Synthesis constraint commands. A complete mapping of

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

Synopsys commands to BuildGates Synthesis commands is available in [SDC Constraints Support Guide](#)

Synopsys Command	BuildGates Synthesis Command
Create_clock	Set_clock Set_clock_root
Set_input_delay	Set_input_delay
Set_output_delay	Set_external_delay
Set_multicycle_path	Set_cycle_addition
Set_false_path	Set_false_path
Set_propogated_clock	Set_clock_propagation
Max_transition	Slew_time_limit
Max_capacitance	Port_capacitance_limit
Max_fanout	Fanout_load_limit
Set_don't_touch	Set_don't_modify
Set_clock_uncertainty	Set_clock_uncertainty
Set_don't_touch_network	Set_don't_modify -network
Set_don't_use	Set_cell_property -don't_use true
Set_disable_timing	Set_disable_timing
Set_clock_latency	Set_clock_insertion_delay

4. Check your HDL for any Synopsys proprietary DesignWare elements. In most cases, you can replace these with inferences. For example, DW01_add can be replaced with a+b.

There are more tips for converting to BuildGates Synthesis in [SDC Constraints Support Guide](#)

Constraint Translation Example

In this section, you will translate Synopsys constraints to BuildGates constraints. Although we have no Synopsys constraints for this design that are generated with `write_script`, they can be generated using the `write_sdc` command in BuildGates.

Using the `read_dc_script` command, you can read in these Synopsys constraints into the BuildGates environment.

The BuildGates Synthesis `read_dc_script` command is capable of translating output from DC or PT `write_script` (in `dclsh` and `Tcl` formats) or `write_sdc` (available from Synopsys in `Tcl` format) into a BuildGates Synthesis format constraints file.

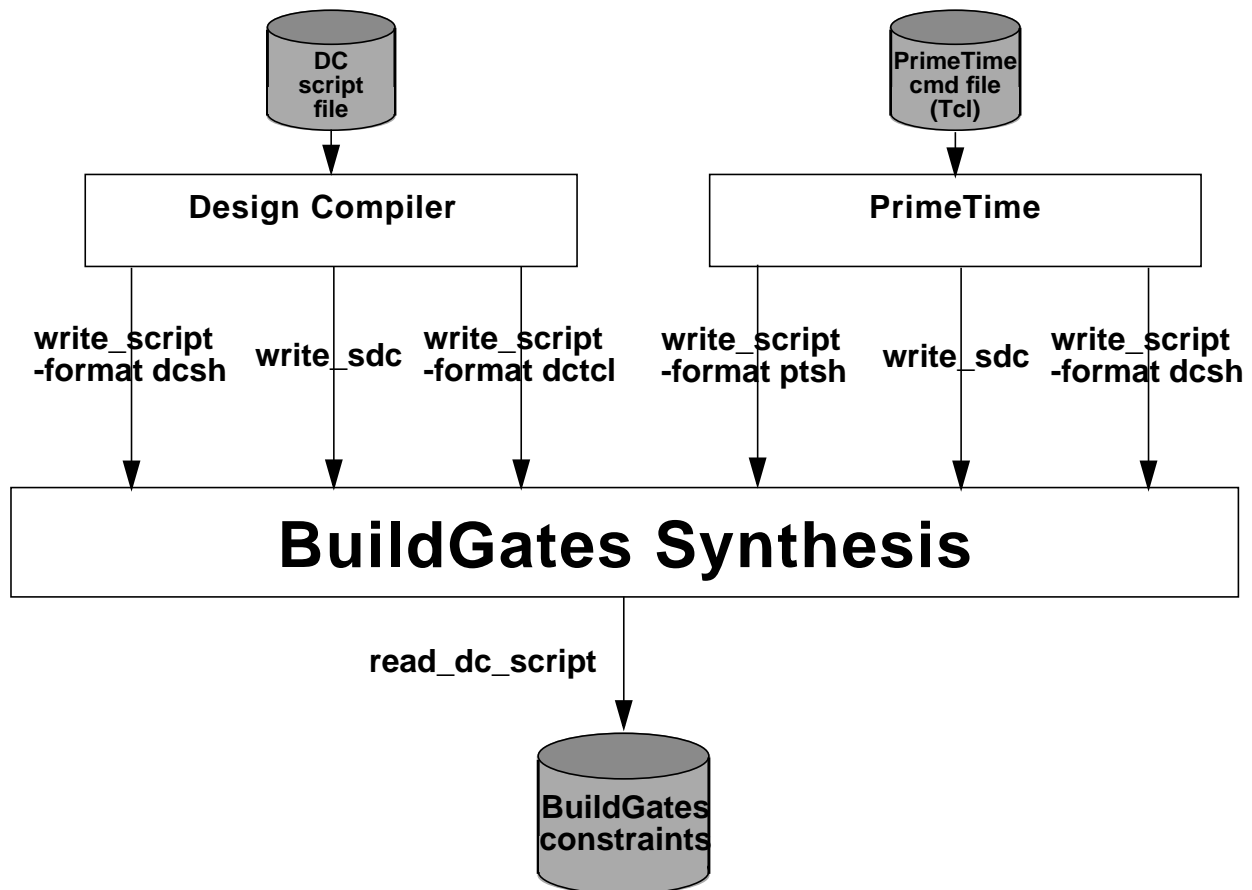
Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

The `read_dc_script` command automatically detects the format of the original constraints file (`dcsh` or `Tcl`) and issues detailed error and warning messages if problems are encountered during translation.

Figure 2-14 shows the translation flow.

Figure 2-14 Synopsys-to-BuildGates Synthesis Translation Flow



1. Write the constraints in Synopsys format:

```
bg_shell > write_sdc sdc_out.tcl
```

The `write_sdc` command writes the constraints in Synopsys constraints format, for use by First Encounter, PrimeTime, and so on. Part of the `sdc_out.tcl` file is shown below.

```
set sdc_version 1.4
current_design dtmf_chip
create_clock [get_ports {refclk}] -name vclk1 -period 6 -waveform {0 3}
create_clock -name vclk2 -period 12 -waveform {0 6}
create_generated_clock -name vclk1_int1 -from [get_pins
{DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/clk}] -divide_by 2 \
[get_pins {DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/write_reg/Q}]
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

```
create_generated_clock -name vclk1_int2 -from [get_pins \
{DTMF_INST/DMA_INST/clk}] -divide_by 2 \
[get_pins {DTMF_INST/DMA_INST/write_reg/Q}]
set_driving_cell -lib_cell PDO04CDG -library slow -no_design_rule -pin PAD\
-from_pin I [get_ports {ibias}]
set_driving_cell -lib_cell PDO04CDG -library slow -no_design_rule -pin PAD\
-from_pin I [get_ports {port_pad_data_in[5]}]
set_driving_cell -lib_cell PDO04CDG -library slow -no_design_rule -pin PAD\
-from_pin I [get_ports {port_pad_data_in[10]}]
set_driving_cell -lib_cell PDO04CDG -library slow -no_design_rule -pin PAD\
-from_pin I [get_ports {reset}]
...
```

Since you do not get any warning or error messages, you can assume that all constraints are translated properly.



Tip

The translator gives warning messages if any constraints are not translated properly. You need to check all the warning messages, before using the constraints. Always review the translated output file quickly. If a constraint is not translated properly, the tool will print the original command with a comment.

2. Read the Synopsys constraints:

```
bg_shell > read_dc_script -ambit bg_constraints.tcl -write_only \
-tcl sdc_out.tcl
```

This command reads a Synopsys constraints file, `sdc_out.tcl`, and writes a file, `bg_constraints.tcl`, in BuildGates constraints format. In this case, you write out the constraints without applying them. You can also apply the constraints on the design loaded in `bg_shell` with the `-apply_only` option. In the latter case, the constraints are not written.

From the log file below, you can see that all synopsys constraints are translated correctly into BuildGates constraint format without any errors or warnings.

Below you find an extract from the log file:

```
Info:      Following commands are not supported and will be suppressed:
<S2A-404>.
Info:      set_local_link_library <S2A-405>.
Info:      converting sdc_out.tcl to tcl format ... <S2A-406>.
Info:      initializing pbc constraints ... <S2A-407>.
Info:      translating constraints ... <S2A-409>.
...
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with BuildGates Synthesis

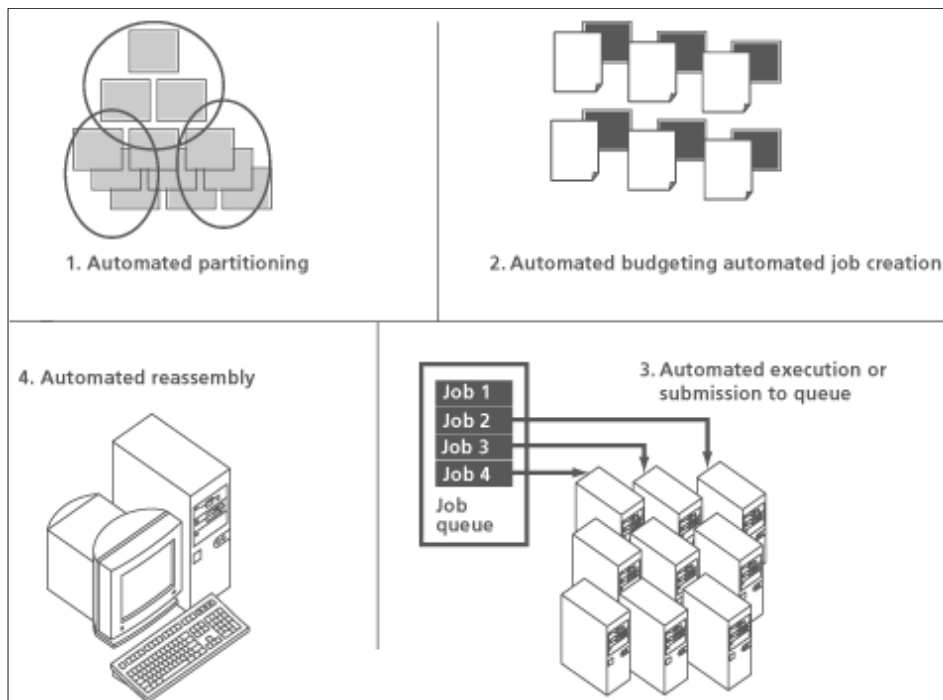
Below you find an extract from the `bg_constraints.tcl` file:

```
### ac_shell version: v5.10-s058 (May 20 2003 13:32:50)
#WARNING: PLEASE DO NOT REMOVE/CHANGE THE FOLLOWING COMMAND - IT SETS THE TIMING
ENGINE IN PT COMPATIBLE MODE
#Removing/Changing this will lead to different interpretation of false path and
multicycle path constraints.
set_flow_compatible_mode on
set_current_module dtmf_chip
set_clock vclk1 -waveform {0 3} -period 6
set_clock_root -clock vclk1 -pos {refclk}
set_clock vclk2 -waveform {0 6} -period 12
...
set_drive_cell -cell {PDO04CDG} -library {slow} -pin {PAD} -from_pin {I}\
-no_design_rule ibias
....
set_constant_for_timing 0 scan_en
set_constant_for_timing 0 test_mode
set_wire_load_mode enclosed
set_wire_load -library slow -pvt max TSMC18_Conservative
...
set_port_capacitance 6.14228 vcop
set_port_capacitance 6.14228 {{port_pad_data_out[6]}}
...
set_slew_time_limit -max 2.3 {{port_pad_data_out[3]}}
set_slew_time_limit -max 2.3 {{port_pad_data_in[12]}}
...
set_input_delay -clock vclk1 -add_delay 0.5 {{port_pad_data_in[6]}}
set_input_delay -clock vclk1 -add_delay 0.5 {{port_pad_data_in[11]}}
...
set_external_delay -clock vclk1 -add_delay 0.5 {{port_pad_data_out[11]}}
set_external_delay -clock vclk1 -add_delay 0.5 {{port_pad_data_out[15]}}
...
```

3. To end the session, select *File – Exit* or type `exit` in the console.

Distributed Synthesis

Distributed Synthesis is a built-in automated parallel processing technology that lets you take advantage of multiple processors working on a single synthesis design, thus achieving significant speedups in run times. This highly automated system, requiring very little direction from you, is a robust system with a built-in recovery mechanism for networking and computer failure. Distributed synthesis also has built-in controls to work seamless in an environment managed by the Platform Computing LSF batch management product.



For more information, refer to [*Distributed Processing for BuildGates Synthesis.*](#)

Summary

Congratulations, you have just finished the introductory module for BuildGates Extreme.

In this session, you learned:

- How BuildGates Synthesis works including test synthesis, timing analysis, translation, mapping and optimization.
- Which commands to use to execute the basic steps in the BuildGates Synthesis flow.
- To use BuildGates Synthesis in both text mode and graphical mode.
- How to migrate from Design Compiler to BuildGates Synthesis.

Where to Go from Here

At this time, you may be interested in completing one of the following tutorials that go into detail on some of the advanced features of BuildGates Extreme:

- [Getting Started with Design For Test](#)
- [Getting Started with Datapath](#) (advanced arithmetic)
- [Getting Started with Low Power Synthesis](#)

Getting Started with Design For Test

This tutorial contains the following sections:

- [Introduction to BuildGates Test Synthesis](#) on page 74
- [Objectives](#) on page 75
- [Example Design](#) on page 75
- [Setting Up to Run the Tutorial](#) on page 77
- [BuildGates Top-Down Test Synthesis Configuration Flow](#) on page 78
 - [Setting up the Design Environment](#) on page 80
 - [Read Libraries and the HDL Models for the Design](#) on page 81
 - [Setting Test Synthesis Assertions](#) on page 82
 - [Checking DFT Assertions and Rule Violations](#) on page 84
 - [Fixing DFT Rule Violations](#) on page 87
 - [Setting Constraints](#) on page 89
 - [Optimizing the Design](#) on page 89
 - [Setting Scan Chain Assertions](#) on page 90
 - [Connecting the Scan Chains](#) on page 92
 - [Checking the Timing of the Design](#) on page 94
 - [Viewing the Scan Chains](#) on page 95
 - [Writing Netlist and Database Files](#) on page 96
- [Writing Interface Files to Third Party ATPG Tools](#) on page 97

Introduction to BuildGates Test Synthesis

Design For Test (DFT) using BuildGates Test Synthesis refers to the set of capabilities of BuildGates® Synthesis to perform test synthesis. Test synthesis consists of a set of automated techniques used to integrate DFT logic into your design.

In BuildGates Synthesis this includes capabilities to automatically map all flip-flops that pass DFT rule checks to their scan-equivalent flip-flops from the target technology library, and configure those flip-flops into scan chains (scan chain configuration) given the presence of clock domain information (output from `check_dft_rules` command). These functions can be applied to designs at the RTL, gate, or mixed level.

Capabilities

- Supports Verilog/VHDL (RTL and/or gate level netlists) formats
- Multiplexed flip flop, clocked-scan, clocked-LSSD, aux-clocked LSSD scan insertion styles
- Module or instance specific `set_dont_scan` attributes
- DFT Rule Checking
- Multiple clock domains
- Multiple balanced scan chains
- Data lockup latch insertion and analysis between compatible clock domains
- Data lockup flip-flop insertion between compatible clock domains
- Configuration of data lockup latches or flops at the end of the scan chains
- Auto-fix of DFT rule violations associated with async set/reset and internal clock nets
- Full-scan test insertion
- Top-down and bottom-up scan insertion
- Test insertion to tieback, chain or floating modes
- Scan chain inversion based on net loading
- Fixed segment handling through `set_dont_touch_scan` attribute
- Shared scan-data input and scan-data output signals
- User-defined scan chain ordering through `read_scan_order_file` command
- Automatic creation of top-level scan-data input and output, and scan mode ports

Objectives

This tutorial is an introduction to performing test synthesis of an RTL design in the top-down flow (shown in [Figure 3-2](#) on page 79) using BuildGates Synthesis.

Upon completion of this tutorial, you will have learned the following:

- Full scan test insertion
- Single-pass test synthesis
- DFT rule checking
- Top-down scan insertion
- Test insertion to tieback and chain mode
- Data lockup latch insertion and analysis between compatible clock domains
- Using existing functional ports for scan-data signals
- Writing ATPG interface files for third-party ATPG tools
- Displaying module level scan chains
- Using `set_dont_modify` and `set_dont_touch_scan` on precompiled lower level modules

For further information on the BuildGates test synthesis features, see the [*Design for Test \(DFT\) Using BuildGates Synthesis and Cadence PKS*](#).

Note: This module assumes that you completed [Chapter 2, “Getting Started with BuildGates Synthesis”](#) and that you are familiar with Design for Test (DFT).

Example Design

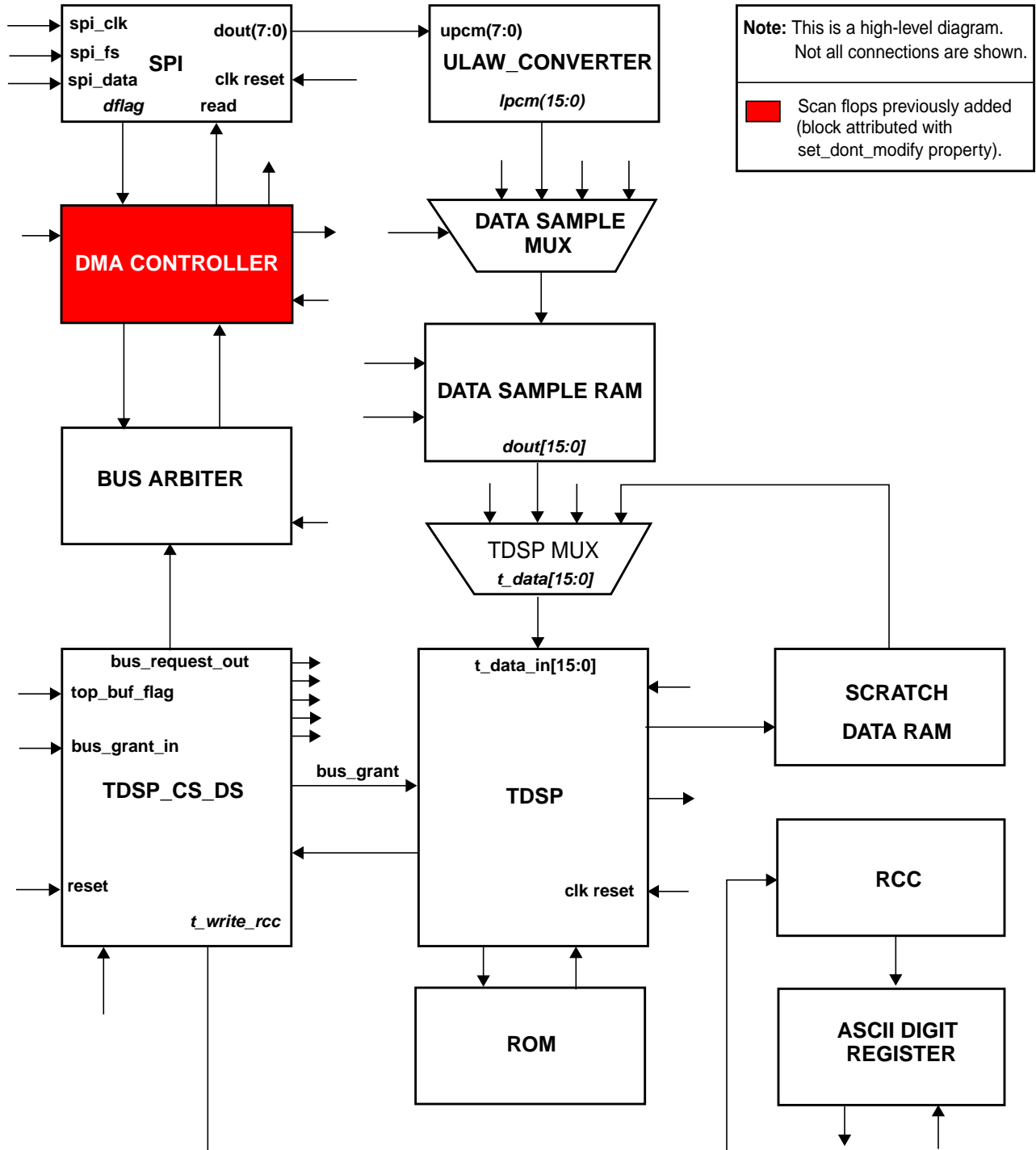
The design in this tutorial is a Verilog description of a Dual Tone Multi-Frequency (DTMF) receiver. In a telephone network, DTMF is a common form of in-band signaling technique used for transmitting information between network entities. DTMF signals are commonly generated by touch tone telephones. [Figure 3-1](#) on page 76 shows the block diagram of DTMF receiver.

The DTMF controller is comprised of RTL modules with the exception of the DMA Controller. This module has previously undergone test synthesis to scan flops. The logic in this module will be preserved during optimization by attributing this block with a `set_dont_modify` property. All other blocks, unless attributed with a `set_dont_scan` property, will undergo test synthesis.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

Figure 3-1 DTMF Block Diagram



Setting Up to Run the Tutorial

Note: We assume that you already copied the data. If not, see [Getting Started](#) on page 12.

Required License

You need a BuildGates (BG) license to run this module.

Required Files

You need the following files to run this tutorial:

tutorial_dft	Root directory
tcl	Scripts to run this module
rtl	Verilog design modules
lib	Synthesis libraries
rpt	Timing, area, hierarchy, dft registers (scan and non-scan), scan order file
adb	Design netlists and database files
run_dir	Running directory

BuildGates Top-Down Test Synthesis Configuration Flow

The top-down test synthesis configuration flow maps all flip-flops which pass the DFT rule checks and which are not attributed with a `set_dont_modify` or `set_dont_scan` property, to their scan equivalent flops and creates scan chains in a single optimization pass.

By default, the number of scan chains configured in the design is a single scan chain for each DFT clock domain identified by the `check_dft_rules` command. You can also create additional scan chains by specifying the `set_number_of_scan_chains` or `set_max_scan_chain_length` assertions, or create merged scan chains using the `set_dft_compatible_clock_domain` assertion.

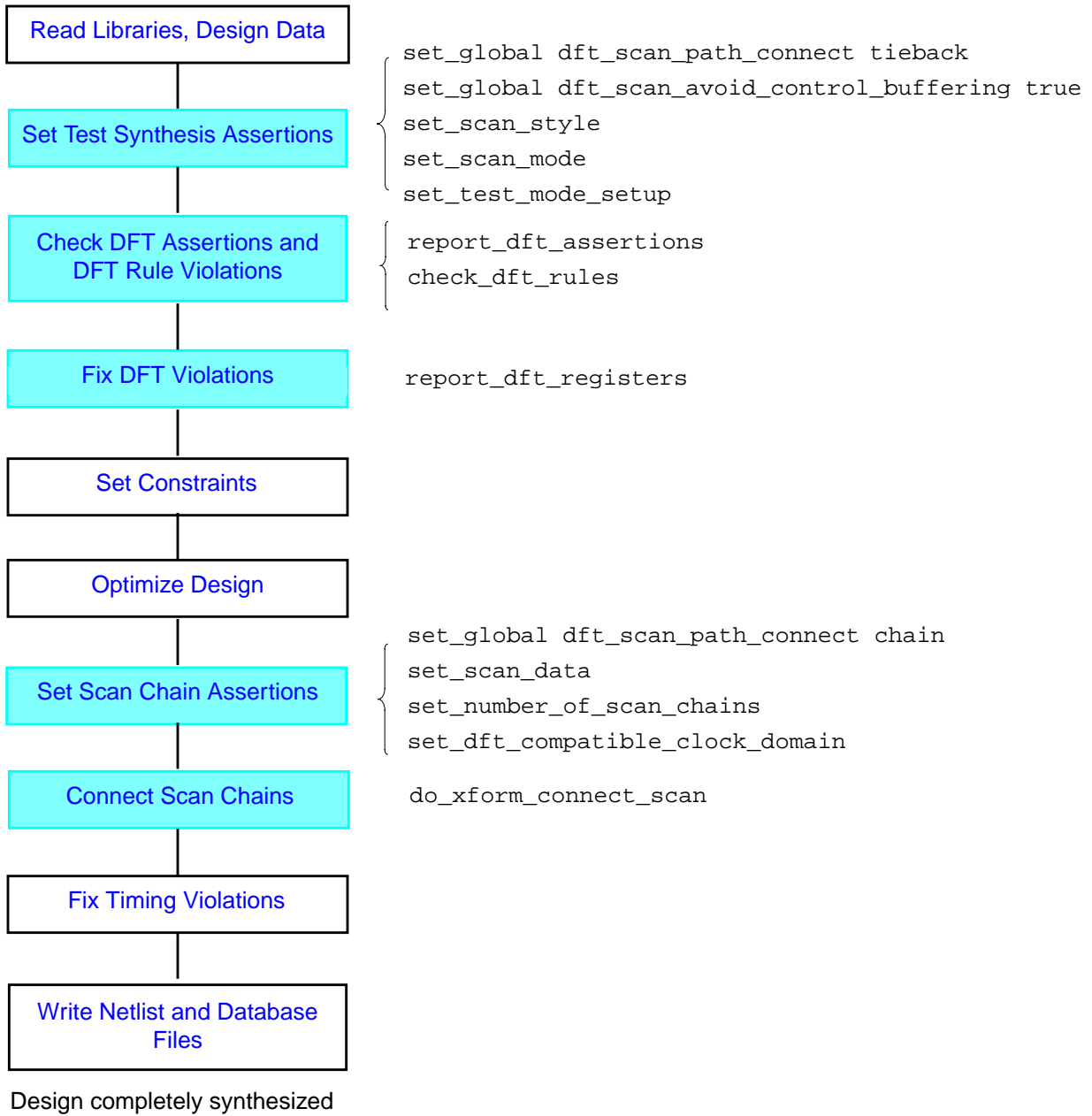
The selection of test synthesis assertions and commands depends on whether you are performing test synthesis in `tieback` mode or creating the scan chain configuration after optimization.

The flow outlined in [Figure 3-2](#) on page 79 is used to synthesize the DTMF controller to scan flops and create the top-level scan chains. [Figure 3-2](#) shows the order in which you specify the test synthesis assertions and commands when the scan flops are being placed in `tieback` mode (see [Setting Test Synthesis Assertions](#) on page 82). After optimization completes, the top-level scan chains are configured by specifying the top-level chain assertions and then running the scan connection engine in chain mode using the `do_xform_connect_scan` command (see [Connecting the Scan Chains](#) on page 92).

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

Figure 3-2 DFT Top-Down Configuration Flow



Note: Colored steps are either DFT-specific or are affected by DFT.

Setting up the Design Environment

Start the Tool

1. Change to the `run_dir` directory in the `tutorial_dft` directory.
2. Invoke BuildGates in Graphical User Interface (GUI) mode.

```
bg_shell -gui &
```

3. Set up your design environment by running the following script in `bg_shell`:

```
bg_shell> source ../tcl/setup.tcl
```

The `setup.tcl` file defines Tcl variables for the rtl, library, report, database, and Tcl directories. This file contains the following commands:

```
set run_dir ./
set rak_dir ../
set tcl_dir $rak_dir/tcl
set rtl_dir $rak_dir/rtl
set lib_dir $rak_dir/lib
set rep_dir $rak_dir/rpt
set adb_dir $rak_dir/adb
```

Set Global Variables

- Set some globals by running the following script:

```
bg_shell>source $tcl_dir/set_globals.tcl
```

The `set_globals.tcl` script sets two variables:

```
set_global message_verbosity_level 3
set_global echo_commands true
```

- The `message_verbosity_level` variable controls the level of verbosity of the messages generated by `bg_shell` in reporting information, warnings, and errors.
- The `echo_commands` variable causes each command to be echoed in the standard output prior to its execution.

Read Libraries and the HDL Models for the Design

1. Read in the synthesis technology libraries by running the following script:

```
bg_shell> source $tcl_dir/read_lib.tcl
```

The `read_lib.tcl` script contains the following statements:

```
read_tlf $lib_dir/slow_4.3.tlf
read_tlf $lib_dir/pllclk_slow_4.3.tlf
read_tlf $lib_dir/ram_128x16A_slow_4.3.tlf
read_tlf $lib_dir/ram_256x16A_slow_4.3.tlf
read_tlf $lib_dir/rom_512x16A_slow_4.3.tlf
read_library_update $lib_dir/tpz973gwc-lite_4.3.tlf
```

```
set_global target_technology slow
```

This script will set the technology libraries and wireload model for this design.

2. View the contents of the technology library with the following command:

```
bgx_shell>report_library
```

3. Read the RTL code for the design by running the following script:

```
bg_shell> source $tcl_dir/read_rtl.tcl
```

The `read_rtl.tcl` script contains the following statements:

```
set_global hdl_verilog_vpp_arg -I$rtl_dir
read_verilog $rtl_dir/accum_stat.v
read_verilog $rtl_dir/alu_32.v
read_verilog $rtl_dir/arb.v
read_verilog $rtl_dir/data_bus_mach.v
read_verilog $rtl_dir/data_sample_mux.v
read_verilog $rtl_dir/decode_i.v
read_verilog $rtl_dir/digit_reg.v
read_verilog $rtl_dir/dma.v
read_verilog $rtl_dir/dtmf_recvr_core.v
read_verilog $rtl_dir/execute_i.v
read_verilog $rtl_dir/mult_32_dp.v
read_verilog $rtl_dir/m16x16.v
read_verilog $rtl_dir/port_bus_mach.v
read_verilog $rtl_dir/prog_bus_mach.v
read_verilog $rtl_dir/ram_128x16_test.v
read_verilog $rtl_dir/ram_256x16_test.v
read_verilog $rtl_dir/results_conv.v
read_verilog $rtl_dir/spi.v
read_verilog $rtl_dir/tdsp_core.v
read_verilog $rtl_dir/tdsp_core_glue.v
read_verilog $rtl_dir/tdsp_core_mach.v
read_verilog $rtl_dir/tdsp_data_mux.v
read_verilog $rtl_dir/tdsp_ds_cs.v
read_verilog $rtl_dir/test_control.v
read_verilog $rtl_dir/ulaw_lin_conv.v
read_verilog $rtl_dir/iopads.v
read_verilog $rtl_dir/dtmf_chip.v
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

4. Transform the design into a hierarchical, gate-level netlist by entering the following command:

```
bg_shell> do_build_generic
```

The `do_build_generic` command transforms the design into a hierarchical, gate-level netlist consisting of technology-independent logic gates. It also performs constant propagation, loop unrolling, lifetime analysis, and register inferencing. You see the following messages:

```
Info:    Setting 'dtmf_chip' as the top of the design hierarchy <FNP-704>.
Info:    Setting 'dtmf_chip' as the default top timing module <FNP-705>.
```

These messages indicate that the top module is set to `dtmf_chip` for logic optimization and scan chain configuration. The top timing module is set to `dtmf_chip` for timing assertions and DFT rule checking. These modules can also be specified by the `set_current_module` and `set_top_timing_module` commands, respectively.

Setting Test Synthesis Assertions

Assertions can be thought of as requirements or guidelines that govern how test synthesis will process the design. Assertions are used by test synthesis commands, such as `check_dft_rules`, `do_xform_connect_scan`, and `write_scan_order_file`.

1. Set test synthesis assertions by running the following script:

```
bg_shell> source $tcl_dir/scan_assert.tcl
```

The script contains two sections.

2. Examine the commands in the first section of the `scan_assert.tcl` file.

The “Mapping test_control module” section of `scan_assert.tcl` file contains the following commands:

```
set_current_module test_control
do_xform_map
set_dont_modify [find -hier -module test_control]
set_current_module [find -module dtmf_chip]
```

The `test_control` module consists of combinational gates which multiplex between the functional clocks (system mode), and the test clock (test mode). The select enable signal to the mux is controlled by the `test_mode` signal.

- The `set_current_module` command sets `test_control` as the design context.
- The `do_xform_map` transform maps the `test_control` module to target technology `slow`.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

- ❑ The `set_dont_modify` command keeps the instances, nets, or submodules in the `test_control` module from being modified by the optimization and test synthesis tool.
- ❑ The second `set_current_module` sets the timing and optimization context to the `dtmf_chip` module.

3. Examine the commands in the second section of the `scan_assert.tcl` file.

The “Setting Test Synthesis Assertions” section of the `scan_assert.tcl` file contains the following commands:

```
set_scan_style muxscan
set_global dft_scan_path_connect tieback
set_global dft_scan_avoid_control_buffering true

set_scan_mode IOPADS_INST/Pscanenip/C 1

set_dont_modify [find -instance DTMF_INST/DMA_INST]
set_dont_touch_scan dma
```

- ❑ The `set_scan_style muxscan` command sets the scan style to `muxscan`. The multiplexed scan style (`muxscan`) is the most commonly used scan style. Test synthesis supports four scan styles: `muxscan`, `clocked_scan`, `clocked_LSSD`, and `aux_clocked_LSSD`. When no style is selected, the default is `muxscan`.
- ❑ The `set_global dft_scan_path_connect tieback` command sets the scan connection mode to `tieback` by test synthesis during optimization. The `tieback` mode connects the scan flop's scan-data output pin to its own scan-data input pin. Cadence recommends that you set this variable to `tieback` before optimization and set the variable to `chain` after optimization, when the scan chain is created and connected.
- ❑ The `set_global dft_scan_avoid_control_buffering true` command is optional. It prevents buffering of the high fanout scan mode control signal after it is globally routed by test synthesis during optimization. By default, high fanout signals are buffered by timing optimization and design rule fixing. This command overrides the default behavior for the scan mode signal only.
- ❑ In the `muxscan` style, the `set_scan_mode IOPADS_INST/Pscanenip/C 1` command specifies an internal pin, `IOPADS_INST/Pscanenip/C`, and its active value (`1`) which is the scan mode signal, used for shifting scan-data into scan flops. If you do not specify this assertion, test synthesis will create a top-level input port named `BG_scan_enable`, with an active high polarity. Additionally, you can specify the `dft_scan_port_name_prefix` global to specify a different default naming convention.
- ❑ The `set_dont_modify [find -instance DTMF_INST/DMA_INST]` command keeps the instances, nets, or modules in `DMA_INST` module from being modified by the optimization tool. The command is needed as the `DMA_INST`

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

module is a precompiled module from a previous synthesis run. This attribute also prevents the scan chains and scan chain order from being modified by the test synthesis tool. If optimization were allowed in the module, but the scan chain configuration was to be preserved, then use the `set_dont_touch_scan` assertion instead. You should specify this assertion prior to creating the top-level scan chains.

Checking DFT Assertions and Rule Violations

The following procedure verifies the test synthesis assertions applied to the database and checks the generic database for DFT rule violations.

1. Display the current test synthesis assertions (optional):

```
bg_shell> report_dft_assertions
```

The `report_dft_assertions` command displays the following test synthesis assertions that are in effect.

```
Info: Scan mode for module 'dtmf_chip' set to 'IOPADS_INST./Pscanenip/C = 1'.
Info: Default Scan mode for module 'dtmf_chip' set to 'IOPADS_INST/Pscanenip/C = 1'.
Info: Scan style for module 'dtmf_chip' set to 'mux_scan'.
Info: module 'dma' set to DONT_TOUCH_SCAN mode.
```

Note: Check for combinational loops and race conditions in your design (optional):

```
set_global dft_enable_combinational_loop_check true
set_global dft_enable_race_condition_check true
```

Setting the above global variables to `true` forces a subsequent execution of `check_dft_rules` to check for and report any combinational feedback loops and any flip-flops with potential race conditions between the data and clock signals.

2. Check for DFT rules violations.

```
bg_shell> check_dft_rules
```

The `check_dft_rules` command checks for DFT rules violations, such as uncontrollable clocks and uncontrollable asynchronous signals. It also checks to make sure clock pins to the flip-flops are directly controllable from a primary input, and that any asynchronous control pins (set/reset) to the flip-flops can be held to their inactive state during the scan shift-cycle of test mode.

The tool does not insert scannable flip-flops where DFT violations occur. Only those flops which pass the DFT rule checks will be mapped onto their scan equivalent flop and placed on the scan chain. The `check_dft_rules` command displays a message if it detects any violations. You should fix all reported problems either by modifying your RTL or by using the auto-DFT fix capabilities described in [Improving Testability of Your Design in Design For Test \(DFT\) Using BuildGates Synthesis and Cadence PKS](#). Otherwise, your fault coverage is reduced.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

The `check_dft_rules` command lists the DFT clock domains, any internal clock domains specified with the `set_dft_internal_clock_domain` assertion, the number of flip-flops associated with those domains that have passed the DFT rule checks, and the number of flip-flops which failed the DFT rule checks. Part of the TDR output is shown below.

```
...
Info:    Checking for registers with uncontrollable clock ports. <DFT-315>.
==> ERROR:  DFT Violation 1: Internally driven Clock net 'DTMF_INST/
TDSP_DS_CS_INST/m_sel_ds_address' in module 'tdsp_ds_cs' (File ../rtl/
tdsp_ds_cs.v, Line 125) <DFT-316>.
    Traced its effective fanin cone to:
        :test_mode
        DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/write_reg:Q
        DTMF_INST/TDSP_CORE_INST/PORT_BUS_MACH_INST/as_reg:Q
        DTMF_INST/TDSP_CORE_INST/DECODE_INST/ir_reg_9:Q
        DTMF_INST/TDSP_CORE_INST/DECODE_INST/ir_reg_10:Q
        :scan_clk
        refclk
        ibias
        pllrst
    . . . . .
==> ERROR:  DFT Violation 3: Internally driven Clock net 'DTMF_INST/m_clk' in
module 'dtmf_recvr_core' (File ../rtl/dtmf_recvr_core.v, Line 84) <DFT-316>.
    Traced its effective fanin cone to:
        :test_mode
        scan_clk
        refclk
        ibias
        pllrst
    . . . . .
Clock pins for dft_top_module dtmf_chip are:
Info:    Checking for registers with uncontrollable Async. Set/Reset ports.
<DFT310>.
Info:    Partitioning registers for scan based on clock domain. <DFT-325>.
Total Clock domains: 0 for 0 f/f
Info:    Total Scannable register count: 0 <DFT-340>.
--> WARNING: Scan mapped register 'DTMF_INST/DMA_INST/breq_reg' in module
'netlist' has a TDR violation: internal clock signal <DFT-334>.
--> WARNING: Scan mapped register 'DTMF_INST/DMA_INST/as_reg' in module
'netlist' has a TDR violation: internal clock signal <DFT-334>.
. . . . .
--> WARNING: Scan mapped register 'DTMF_INST/DMA_INST/a_reg_1' in module
'netlist' has a TDR violation: internal clock signal <DFT-334>.
--> WARNING: Scan mapped register 'DTMF_INST/DMA_INST/a_reg_0' in module
'netlist' has a TDR violation: internal clock signal <DFT-334>.
Info:    Checking for available scan cells in Library. <DFT-330>.
--> WARNING: Design has 6 TDR violations <DFT-342>.
```

If a flip-flop fails the DFT rule checks, a fanin trace from the source of the violation to the top-level ports will be automatically performed. The fanin trace attempts to identify all top-level primary input ports associated with the cone of logic originating from the failure.

The goal is to identify a primary input port which when placed in test mode, will bypass the DFT rule violation. Of course, this presumes that the bypass logic has already been coded in the RTL.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

Following is one of the errors reported by the `check_dft_rules` command.

```
==> ERROR: DFT Violation 6: Internally driven Clock net 'DTMF_INST/  
m_digit_clk' in module 'dtmf_recvr_core' (File ../rtl/dtmf_recvr_core.v, Line  
84) <DFT-316>.
```

```
Traced its effective fanin cone to:
```

```
:test_mode
```

```
scan_clk
```

```
DTMF_INST/RESULTS_CONV_INST/digit_clk_reg:Q
```

The above fanin trace indicates that the DFT rule violation maybe possibly bypassed by setting the top-level signal, test mode, to an appropriate logic value using the `set_test_mode_setup` assertion. The value of the assertion is propagated through the circuit on the next invocation of `check_dft_rules` command.

Fixing DFT Rule Violations

1. Fix the DFT rule violations by running the following script:

```
bg_shell> source $tcl_dir/fix_dft_rules.tcl
```

The `fix_dft_rules.tcl` script has the following commands:

```
set_test_mode_setup test_mode 1
set_test_mode_setup spi_fs 0
```

These assertions set the `test_mode` signal to a logic 1, and set the asynchronous set or reset signal `spi_fs` to a logic 0 for the duration of test mode session. When specified, the following Info messages are displayed:

```
set_test_mode_setup test_mode 1
Info: Test mode for module 'dtmf_chip' set to 'test_mode = 1' during entire
test session.
set_test_mode_setup spi_fs 0
Info: Test mode for module 'dtmf_chip' set to 'spi_fs = 0' during entire test
session.
```

Additionally, if an asynchronous set or reset signal had been directly controllable from a primary input port, such that this signal fanned out directly (or through buffers or inverters) to the asynchronous set/reset pins of the flip-flops, the `check_dft_rules` command would have automatically asserted the test mode setup for this signal. The value asserted for the signal is specified to maximize the number of flip-flops which pass the DFT rule checks. When the assertion is automatically specified, the `check_dft_rules` command displays the following message:

```
TDRC identified the following PI ports as active set/reset control pins to be
held constant during scan-shift operation:
Set PI port reset to logic 0 during scan-shift
```

Note: This assertion can be manually specified as follows:

```
bg_shell> set_test_mode_setup reset 0 -scan_shift
```

2. Having specified the test mode setup signals, re-run the DFT rule checks to propagate the logic values of these signals through the circuit.

```
bg_shell> check_dft_rules
```

The output from `check_dft_rules`, also known as DFT rule check data, is used by the test synthesis tool to configure the scan chains during optimization. The final output from the `check_dft_rules` command is shown below:

```
Info: Checking for registers with uncontrollable clock ports. <DFT-315>.
Clock pins for dft_top_module dtmf_chip are: scan_clk
Info: Checking for registers with uncontrollable Async. Set/Reset ports.
<DFT-310>.
...
Info: Partitioning registers for scan based on clock domain. <DFT-325>.
Clock Domain 0 from pin 'scan_clk' (Neg Edge) <Internal Pin: 'None'> has 129 f/f
Clock Domain 1 from pin 'scan_clk' (Pos Edge) <Internal Pin: 'None'> has 411 f/f
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

```
Total Clock domains: 2 for 540 f/f
Info: Total Scannable register count: 540 <DFT-340>.
Info: Checking for available scan cells in Library. <DFT-330>.
Info: Design has no TDRC violation <DFT-343>.
```

At this stage, there are no DFT rule check violations. The DFT rule check report indicates the top-level clock pins, their DFT clock domains, and the number of flip-flops in each domain which pass the DFT rule checks. You can see that both edges of the `scan_clk` are used in the design for a total flip-flop count of 540.

In specific:

- ❑ DFT Clock domain 0 is associated with 129 flip-flops, which are negative edge triggered.
- ❑ DFT Clock domain 1 is associated with 411 flip-flops, which are positive edge triggered.

In the absence of specifying configuration assertions such as `set_number_of_scan_chains`, `set_max_scan_chain_length`, or `set_dft_compatible_clock_domains`, test synthesis would use the DFT rule check data to create a default configuration. The default configuration creates a single scan chain for each DFT Clock domain. In this design, test synthesis would create two scan chains, one of length 411 comprising the positive edge-triggered flops and one of length 129, comprising the negative edge-triggered flops.

3. Generate a report for the list of scannable flops.

```
bg_shell> report_dft_registers -scan > $rep_dir/dft_registers.scan
```

The `report_dft_registers -scan` command writes scan flops identified by `check_dft_rules` to the file `../rpt/dft_registers.scan`. The tool reports the following message:

```
Info: Total Scannable register count: 540 <DFT-340>.
```

4. Generate a report for the list of non-scannable flops.

```
bg_shell> report_dft_registers -non_scan > $rep_dir/dft_registers.nonscan
```

The `report_dft_registers -non_scan` command writes nonscan flops identified by `check_dft_rules` to the file `../rpt/dft_registers.nonscan`. The tool reports the following message:

```
Info: Total Non-Scannable register count: 0 <DFT-341>.
```

Flip-flops, which will be placed in the non-scan list, are those flip-flops that fail the DFT rule checks, or are attributed with a user-specified `set_dont_scan` property.

Since there are no DFT rule check violations, you can now proceed with specifying the timing assertions prior to performing test synthesis (mapping the flip-flops to their scan equivalent flops) and optimization on the design.

Setting Constraints

- Set the constraints by running the following script:

```
bg_shell>source $tcl_dir/constraints.tcl
```

The `constraints.tcl` file sets up timing constraints for the `dtmf_chip` design. For a detailed description of the constraints, see [Setting Constraints](#) on page 30.

Optimizing the Design

- Optimize the DTMF design using the top-down synthesis methodology by entering the following command:

```
bg_shell>do_optimize
```

The `do_optimize` command does the following:

- Generic optimization
- Technology mapping to scan flops (for all flip-flops which pass the DFT rule checks)
- Globally routes the scan mode control signal, as per the `dft_scan_enable_connect` global
- Connects the scan flops as per the `dft_scan_path_connect` global
- Logic optimization

Using the DFT rules check data, the test synthesis tool determines the scan status of flip-flops and to which DFT clock domain the flip-flops belong. In addition, the tool adds a scan mode port (tieback and chain mode), and it adds pairs of scan-data in and scan-data out ports (chain mode only), if required.

The test synthesis tool maps the design's flip-flops to scan flops and connects the scan chains as specified by the `dft_scan_path_connect` global variable. Because you set `dft_scan_path_connect` to `tieback`, the tool connects the scan-data output pin of each scan flop back to its own scan-data input pin.

Note: Depending on the performance of your machine, the optimization runs for a while.

Setting Scan Chain Assertions

The test synthesis tool was run in tieback mode, which connects the scan flop's scan-data output pin to its own scan-data input pin. Tieback mode speeds up synthesis because chain path connections are not being considered during optimization.

- Now that optimization is completed, set the scan chain assertions and connect (configure) the scan chain in the current module (`dtmf_chip`).

```
bg_shell> source $tcl_dir/scan_chain_assert.tcl
```

The `scan_chain_assert.tcl` script contains the following commands:

```
set_number_of_scan_chains 8
set_dft_compatible_clock_domain -same_clock

set_scan_data {IOPADS_INST/Ptdspip00/C} {IOPADS_INST/Ptdspop00/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip01/C} {IOPADS_INST/Ptdspop01/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip02/C} {IOPADS_INST/Ptdspop02/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip03/C} {IOPADS_INST/Ptdspop03/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip04/C} {IOPADS_INST/Ptdspop04/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip05/C} {IOPADS_INST/Ptdspop05/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip06/C} {IOPADS_INST/Ptdspop06/I} -shared_out
set_scan_data {IOPADS_INST/Ptdspip07/C} {IOPADS_INST/Ptdspop07/I} -shared_out
```

- The `set_number_of_scan_chains 8` assertion specifies the total number of top-level scan chains which are to be configured across all DFT clock domains identified by the `check_dft_rules` command. This assertion will create balanced scan chains.

Note: In addition to the above assertions, you can specify the scan chain length using the `set_max_scan_chain_length` assertion. This assertion will pack the scan chains up to the limit specified, with any residual scan flops placed on a new chain. Both the `set_number_of_scan_chains` and `set_max_scan_chain_length` assertions can be optionally specified with `-clock` (`-rise` and `-fall` options) which allows for clock domain-specific and clock edge-specific configuration of the scan flops which pass the DFT rule checks.

- The `set_dft_compatible_clock_domain` assertion specifies the domain and phase compatibility between the DFT clock domains. This assertion optionally instructs test synthesis to insert a data lockup element between flip-flops belonging to the scan chain segments from the different DFT clock domains.

The `-same_clock` option instructs test synthesis to combine alternate edges of the same root clock in a single scan chain. If the DFT clock waveform is set to Return-to-Zero (RTZ), the negative edge-triggered segments proceed the positive edge-triggered segments in the same scan chain avoiding the need to insert a data lockup element between the domain transitions. If however, the `set_dft_clock_waveform` is set to Return-to-One (RT1), the positive edge-triggered segments proceed the negative edge-triggered segments in the same scan chain avoiding the insertion of a data lockup element between the domain transitions. The `set_dft_clock_waveform` assertion

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

must be specified prior to running the `check_dft_rules` command. The default value for this assertion is `RTZ`. For this example, the root clock domain is `scan_clk`. The message from this assertion is excerpted below:

```
Info: Compatible clock mode set to 'DOMAINS THAT ARE PHASES OF SAME CLOCK CAN BE MERGED'.
```

- The `set_scan_data` assertion specifies the names for the scan-data input and scan-data output signals which are used to shift data into and out of the scan chains, respectively. The scan-data signals may be specified to top-level ports or hierarchical pins. The scan-data input signal can be a dedicated port for scan in purposes only, or it can be a shared functional input port. The scan-data output port can be a dedicated port for scan out purposes only, or it can be a shared functional output port. Use the `-shared_out` option if the scan-data output signal is also a shared output. This option instructs test synthesis to insert a multiplexer to select between the internal scan-data output (from the last scan flop in the scan chain), and the internal functional output signal. The select control to the multiplexer is the scan mode signal, declared using the `set_scan_mode` assertion.

For the eight sets of scan-data assertions above, the scan-data signals have been specified to existing hierarchical pins. When the scan chains are configured in the design, the connection engine uses these pins to create the connections from the scan-data input signals to the scan-data input pin of the first scan flop in each scan chain, and from the last scan flop output pin in each scan chain, multiplexed with the internal functional output signal to the scan-data output signal. The multiplexer is inserted just before the scan-data output signal, in the `iopads` module.

When specified, the following Info messages are displayed:

```
Info: Existing scan input port 'IOPADS_INST/Ptdspip00/C' will be used for scan insertion.
Info: Existing scan output port 'IOPADS_INST/Ptdspop00/I' will be used for scan insertion.
Info: Scan data I/O for module 'dtmf_chip' set to:
<no clock domain>: {IOPADS_INST/Ptdspip00/C IOPADS_INST/Ptdspop00/I }.
```

where

`IOPADS_INST/Ptdspip00/C` is the output pin of input pad, `port_pad_data_in`
`IOPADS_INST/Ptdspop00/I` is the input pin of the output pad, `port_pad_data_out`

Connecting the Scan Chains

Before optimization, the global variable `dft_scan_path_connect` was set to `tieback` mode. You will now set this variable to `chain` mode to create the scan chains in the design.

1. Specify to connect the scan chain in `chain` mode:

```
bg_shell> set_global dft_scan_path_connect chain
```

When the scan chain connection engine is run, test synthesis creates the scan chain configuration by connecting the scan-data output pin from one scan flop to the scan-data input pin of the next scan flop in the scan chain.

In `chain` mode, the following two additional DFT globals control how the chains are connected.

- ❑ `dft_scan_output_pref` controls whether to use normal or inverted output (Q or Q-bar) of scan flip-flop for scan connection

The possible settings are:

- `non_inv`: use non-inverted (Q) output
- `min_load`: use the output with least load (DEFAULT)

- ❑ `dft_allow_scan_path_inv` controls if inverters should be inserted in scan path to compensate for logic inversion when connecting from inverted output or to inverted scan-input.

The possible settings are:

- `on`: inversion in scan path is allowed. (DEFAULT)
- `off`: inversion in scan path is not allowed. When connecting from inverted output (Q-bar) or to inverted scan-input, an inverter will be inserted to compensate for logic inversion in the scan data path.

2. Run the scan chain connection engine.

```
bg_shell> do_xform_connect_scan -scan_file dtmf_chip.scan.flat
```

The `do_xform_connect_scan` uses the clock domain information generated by the `check_dft_rules` command (shown below) to connect the scan flops into scan chains and creates a scan chain order file in flat format in the run directory, named `dtmf_chip.scan.flat`.

```
Clock Domain 0 from pin 'scan_clk' (Neg Edge) <Internal Pin: 'None'> has 129 f/f  
Clock Domain 1 from pin 'scan_clk' (Pos Edge) <Internal Pin: 'None'> has 411 f/f
```

If you do not specify a name for the scan chain order file, it will be named the same as the top-level module (`set_current_module`) with an extension of `.scan.flat`.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

Due to the setting of `set_number_of_scan_chains` in this design, eight scan chains have been created for scan flops clocked by `scan_clk`. Test synthesis will create balanced (equal length) scan chains as follows:

- ❑ One scan chain of length 68 is created in Clock Domain 0 for a total scan flop count of 68.
- ❑ Six scan chains of length 68 are created in Clock Domain 1 for a total scan flop count of 408.

Due to the setting of the `set_dft_compatible_clock_domain` to `-same_clock`, one scan chain of length 64 is created between two scan chain segments clocked by the `scan_clk` negedge and `scan_clk` posedge respectively. Using this order no data lockup element is needed between the domain transitions. The scan chain segment in the `scan_clk` negedge domain has three scan flops and the scan chain segment in the `scan_clk` posedge domain has 61 scan flops.

Note: If a data lockup latch would be inserted, it would be denoted using `llatch` notation in the scan chain order file.

When the connection engine is run, it first analyzes the lower level modules in order to identify existing scan chain segments. These segments are then incorporated into the top-level scan chains. Note that the scan chain analysis routine analyzed a 17-scan flop scan chain segment in precompiled module `dma`. Since this module was attributed with `set_dont_modify` and `set_dont_touch_scan` properties, the order of the scan flops in the scan chain segment will remain intact when incorporated into the top-level scan chains.

`do_xform_connect_scan` generates the following messages while running:

```
Collecting scan chains for module ram_128x16_test <DFT-501>.
Collecting scan chains for module spi <DFT-501>.
Collecting scan chains for module arb <DFT-501>.
Collecting scan chains for module dma <DFT-501>.
Analyzed chain # 1 (total 17 regs; BG_scan_in -> BG_scan_out)
connected in module 'dma'. <DFT-025>.
```

When the top-level scan chains are being created by the connection engine, it will also generate an Info message to indicate the insertion of data lockup latches.

```
Top-level chain 1 (IOPADS_INST/Ptdspip00/C -> IOPADS_INST/Ptdspop00/I) has 68
registers. <DFT-024>.
Top-level chain 2 (IOPADS_INST/Ptdspip01/C -> IOPADS_INST/Ptdspop01/I) has 68
registers. <DFT-024>.
Top-level chain 3 (IOPADS_INST/Ptdspip02/C -> IOPADS_INST/Ptdspop02/I) has 68
registers. <DFT-024>.
Top-level chain 4 (IOPADS_INST/Ptdspip03/C -> IOPADS_INST/Ptdspop03/I) has 68
registers. <DFT-024>.
Top-level chain 5 (IOPADS_INST/Ptdspip04/C -> IOPADS_INST/Ptdspop04/I) has 68
registers. <DFT-024>.
Top-level chain 6 (IOPADS_INST/Ptdspip05/C -> IOPADS_INST/Ptdspop05/I) has 68
registers. <DFT-024>.
```

Cadence Synthesis Rapid Adoption Kit Getting Started with Design For Test

Top-level chain 7 (IOPADS_INST/Ptdspip06/C -> IOPADS_INST/Ptdspop06/I) has 68 registers. <DFT-024>.
Top-level chain 8 (IOPADS_INST/Ptdspip07/C -> IOPADS_INST/Ptdspop07/I) has 64 registers. <DFT-024>.

Finally, test synthesis will report the top-level scan chains which have been configured in the design.

```
Collecting scan chains for module dtmf_chip <DFT-501>.
Analyzed chain # 1 (total 68 regs; IOPADS_INST/Ptdspip00/C ->
IOPADS_INST/Ptdspop00/I) connected in module 'dtmf_chip'.
<DFT-025>.
Analyzed chain # 2 (total 68 regs; IOPADS_INST/Ptdspip01/C ->
IOPADS_INST/Ptdspop01/I) connected in module 'dtmf_chip'.
<DFT-025>.
...
Analyzed chain # 7 (total 68 regs; IOPADS_INST/Ptdspip06/C -> IOPADS_INST/
Ptdspop06/I) connected in module 'dtmf_chip'. <DFT-025>.
Analyzed chain # 8 (total 64 regs; IOPADS_INST/Ptdspip07/C -> IOPADS_INST/
Ptdspop07/I) connected in module 'dtmf_chip'. <DFT-025>.
Finished scan connection <DFT-400>.
```

This report shows how the scan chains are evenly balanced with equal scan chain lengths using BuildGates Synthesis.

Checking the Timing of the Design

- Check the timing.

```
bg_shell> report_timing > $rep_dir/time_scan.rpt
```

The `time_scan.rpt` shows that timing is met for this design.

Important

If there are any timing violations after scan chain connection, you need to perform incremental timing optimization.

Viewing the Scan Chains

The design contains eight scan chains since the `set_number_of_scan_chains` assertion is set to 8. You can examine the scan chains in a text file or in the schematic viewer.

1. Examine the flat scan chain report file.

```
more dtmf_chip.scan.flat
```

The flat scan chain report file is organized by scan chain, listing all the bits on each scan chain in the order of connection.

A sample of the `dtmf_chip.scan.flat` file is shown below:

```
module dtmf_chip
begin_chain 1 port IOPADS_INST/Ptdspip00/C
bit      1      DTMF_INST/DIGIT_REG_INST/flag_out_reg/Q
bit      2      DTMF_INST/DIGIT_REG_INST/digit_out_reg_7/Q
bit      3      DTMF_INST/DIGIT_REG_INST/digit_out_reg_6/Q
. . . . .
. . . . .
bit      66     DTMF_INST/SPI_INST/bit_cnt_reg_2/Q
bit      67     DTMF_INST/SPI_INST/bit_cnt_reg_1/Q
bit      68     DTMF_INST/SPI_INST/bit_cnt_reg_0/Q
end_chain 1 port IOPADS_INST/Ptdspop00/I <shared_or_complex_out>
```

2. Display the scan chains with the schematic viewer.

- a. Select *Edit – Preferences – Schematic*.

The Schematic Preferences form appears.

- b. Enable *Display scan chain*.

- c. Click OK.

- d. Double-click on the top module.

The GUI highlights the scan chains in the design as shown in [Figure 3-3](#) on page 96.

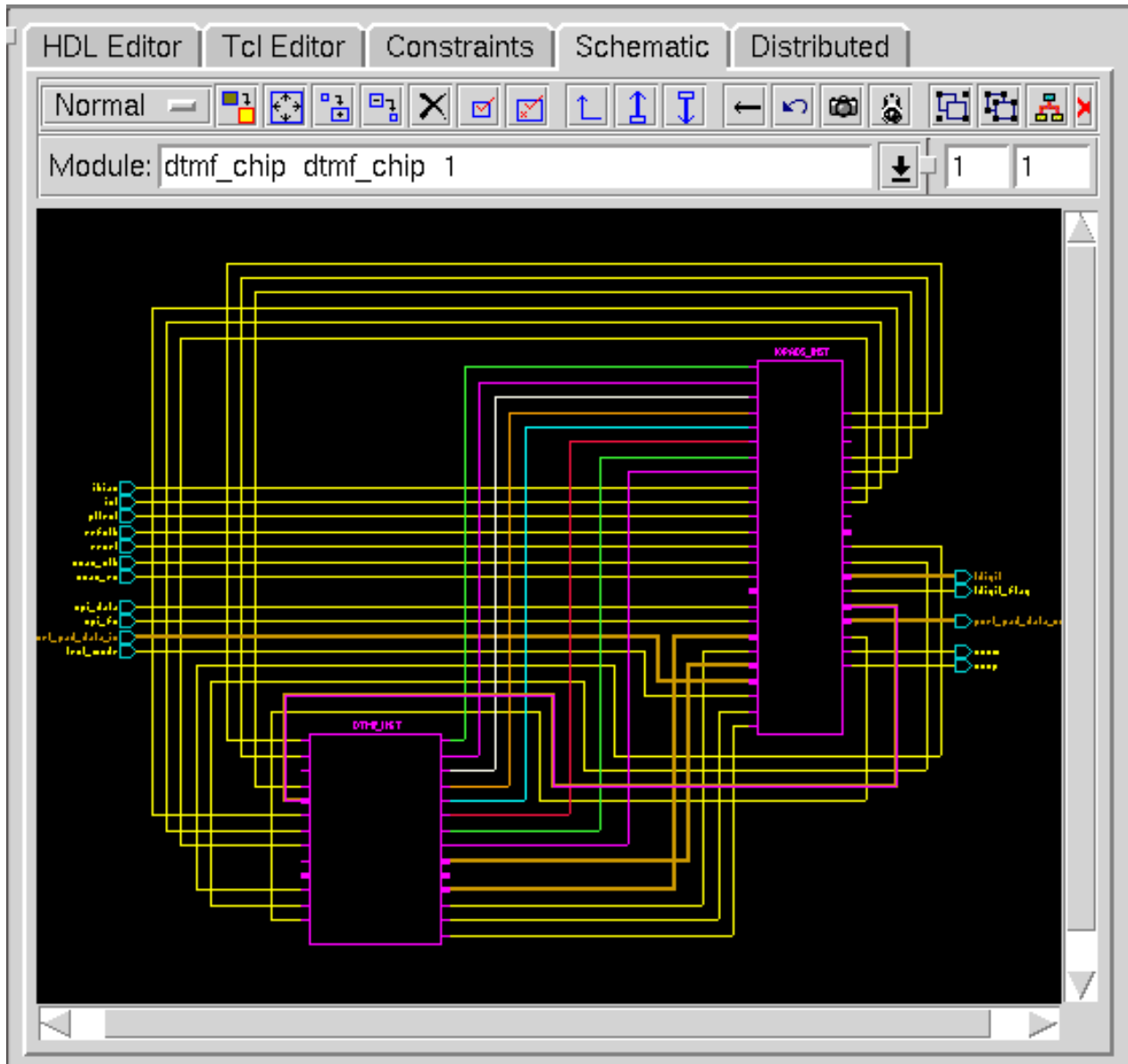
Note: You can also view the scan chains as follows:

- e. Double-click on the top module to display the schematic in the Schematic window.

- f. Display the scan chains for the top module:

```
display_scan_chains [get_names [get_current_module]]
```

Figure 3-3 Scan Chains Displayed in Schematic Viewer



Writing Netlist and Database Files

Optimization is complete and the scan chains are connected.

- Write the netlist and database:

```
bg_shell> write_verilog -hierarchical $adb_dir/dtmf_chip_scan.vs  
bg_shell> write_adb $adb_dir/dtmf_chip_scan.adb
```

Writing Interface Files to Third Party ATPG Tools

BuildGates generates interface files containing scan chain information in the IEEE Standard Test Interface Language (STIL) format or in a format readable by third party ATPG tools like Mentor FastScan[®], Syntest Turboscan[®], and LogicVision[®] ATPG tools.

1. Remove `scan_mode` and `scan_data` DFT assertions, which are set on the internal pad pins and set them with top-level ports.

```
bg_shell> source $tcl_dir/remove_dft_assertions.tcl
```

The `remove_dft_assertions.tcl` script contains the following commands:

```
remove_dft_assertions -scan_mode -scan_data_io

#set top level scan port assertions

set_scan_mode scan_en 1

set_scan_data {port_pad_data_in[7]} {port_pad_data_out[7]} -shared_out
set_scan_data {port_pad_data_in[6]} {port_pad_data_out[6]} -shared_out
set_scan_data {port_pad_data_in[5]} {port_pad_data_out[5]} -shared_out
set_scan_data {port_pad_data_in[4]} {port_pad_data_out[4]} -shared_out
set_scan_data {port_pad_data_in[3]} {port_pad_data_out[3]} -shared_out
set_scan_data {port_pad_data_in[2]} {port_pad_data_out[2]} -shared_out
set_scan_data {port_pad_data_in[1]} {port_pad_data_out[1]} -shared_out
set_scan_data {port_pad_data_in[0]} {port_pad_data_out[0]} -shared_out
```

The `remove_dft_assertions` command removes DFT assertions that were set using `set_scan_mode` and `set_scan_data` assertions. Since the assertions were specified to internal pins (pads), for ATPG purpose these are removed and re-specified using top-level ports.

The eight `set_scan_data` commands are used to specify the top-level ports to be used for generation of ATPG vectors. Now the platform is set for generation of ATPG interface files for the third-party ATPG tools.

2. Write ATPG interface files.

- For FastScan, enter

```
bg_shell> write_atpg_info -mentor
```

This command writes `dtmf_chip.dofile` (command file) and `dtmf_chip.testproc` (test procedure) files into the run directory.

- For Syntest, enter:

```
write_atpg_info -syntest
```

- For LogicVision, enter:

```
write_atpg_info -logicvision
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with Design For Test

- For an interface file in the IEEE Standard Test Interface Language (STIL) format, enter:

```
write_atpg_info -stil
```

3. To end the session, select *File – Exit* or type `exit` in the console.

Summary

In this session, you

- Became familiar with the DFT top-down flow which performs test synthesis of an RTL design.
- Learned that DFT automatically maps all flip-flops that pass DFT rule checks to their scan-equivalent flip-flops from the target technology library, and configures those flip-flops into scan chains given the presence of clock domain information
- Learned how to display the scan chains
- Learned how to write ATPG interface files for third-party ATPG tools

Getting Started with Datapath

- [Introduction to BGX Datapath](#) on page 100
- [Objectives](#) on page 101
- [Example Design](#) on page 101
- [Setting Up to Run the Tutorial](#) on page 102
- [Traditional Synthesis Run](#) on page 103
- [BGX Datapath Flow](#) on page 104
 - [Setting up the Design Environment](#) on page 105
 - [Setting up Global Variables](#) on page 105
 - [Reading the Synthesis Technology Libraries](#) on page 106
 - [Reading the HDL Models for the Design](#) on page 106
 - [Creating a Generic Gate-Level Netlist](#) on page 106
 - [Setting Constraints](#) on page 107
 - [Operator Merging](#) on page 108
 - [Optimizing the Design](#) on page 111
 - [Generating Reports](#) on page 111
 - [Writing Netlist, Database Files and Report File](#) on page 114
- [Examining Traditional Synthesis Results](#) on page 115
- [Comparing BGX and Traditional Results](#) on page 115

Introduction to BGX Datapath

Datapaths are multi-bit bus structures consisting of arithmetic components such as adders, multipliers, decoders, counters, Multiply-Accumulators (MAC), and so on. Traditionally, computing and multimedia designs had some datapath contents, but recently with the integration of computing devices and telecom devices, the usage of datapaths has increased. Telecom designs implement filters with many datapath components. In addition, as a result of the market convergence of consumer, communication, computing, and multimedia applications, most IC designs have complex datapath contents.

Traditional logic synthesis tools optimize datapath contents by providing a library consisting of commonly used datapath components. Very often designers have to overwrite these instances by using pragmas or global variables to produce better results. If these components end up on a critical path, each one must be individually optimized to improve timing. In designs that have many datapath components, there are opportunities for improvement by eliminating redundancies in multiply connected components. Many times designers handcraft custom datapath components to eliminate these redundancies and to achieve the desired performance.

Another approach designers use to synthesize high performance datapath designs involves manually partitioning the datapath portion of the design, writing datapath description in non-standard, proprietary language (such as MCL), and optimizing it in a specialized, standalone tool. The following are some disadvantages to these techniques:

- Handcrafted datapath components often require datapath expertise that is not available.
- Specialized datapath solutions are difficult to implement and use.
- Stand-alone datapath synthesis solutions require learning a new language to describe the datapath logic, which results in a loss of portability across tools and limits your ability to reuse designs.
- Timing closure requires manual effort when you use point tools.
- Designs must be manually partitioned.
- You have to manually explore various implementations and choose the best one.
- Quality of Results (QOR) leaves much to be desired.

BuildGates Extreme (BGX) addresses all of these issues. As you will see in this tutorial, BGX delivers hand-crafted quality results. It automatically takes advantage of redundancies to build optimal components and automatically selects architectures for you based on the context of the component. (For more information on the library see [More about the AmbitWare Library](#) on page 163). BGX eliminates the need for a separate point tool or specialized datapath or language knowledge. For more information about BGX datapath features see [*Datapath for BuildGates Synthesis and Cadence PKS*](#).

Objectives

In this tutorial, you will work on a Finite Input Response (FIR) filter design using the datapath feature of the BuildGates Extreme (BGX) flow, and compare the results to those from a traditional flow.

- Learn about BuildGates Extreme (BGX) datapath features.
- Compare the quality of results from BuildGates Extreme with the results from traditional synthesis.

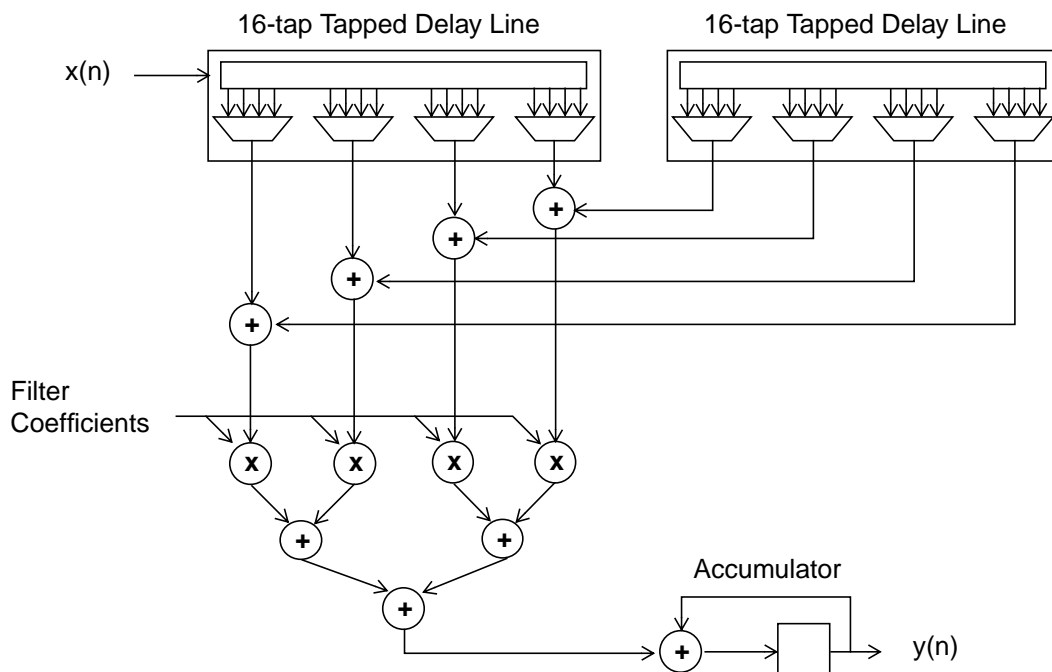
Note: This tutorial assumes that you have already gone through the [Getting Started with BuildGates Synthesis](#) (the basic module of the Rapid Adoption Kit) and are familiar with basic synthesis.

Example Design

You will use an FIR filter (shown in Figure 4-1) with the following specifications:

- 32-TAP
- 16-bit coefficient
- 16-bit I/O data

Figure 4-1 FIR Filter Architecture



Setting Up to Run the Tutorial

Note: We assume that you already copied the data. If not, see [Getting Started](#) on page 12.

Required License

You need a BuildGates Extreme (BGX) license to run this tutorial. The BGX adds Datapath (DP) and Low Power Synthesis (LPS) capabilities to BuildGates Synthesis.

To save running time, you can run `bg_shell` in parallel with `bgx_shell`, one for the regular flow and the other for DP flow. This requires either two BGX licenses (one for running `bg_shell`) or one BG and one BGX license. Call your sales representative to get demo licenses if needed.

Required Files

To run datapath you need the following:

- Datapath content in your design. Specifically, you need to have some of the four basic arithmetic functions (+, -, *, /) or some of the comparison operators (<, >, >=, and so forth). The FIR filter that you will be working with has four multipliers and nine adders.
- Basic synthesis pre-requisites: a properly constrained design and a synthesis library.
- Verilog or VHDL code for the design, preferably at the RTL level. Datapath must be either inferred or instantiated. (Gate-level netlists offer little improvement opportunity). For more information on the supported languages, see [More about Extended Languages](#) on page 166.

The `your_install_path/doc/syntut/RAK/tutorial_dp` directory contains the files needed to run this module and has the following directory structure:

:

<code>tutorial_dp</code>	Root directory
<code>tcl</code>	Scripts to run the RAK DP module
<code>rtl</code>	Verilog design modules
<code>lib</code>	Synthesis library
<code>rpt</code>	Design outputs, including timing, area, hierarchy, and other report files
<code>adb</code>	Design outputs, including netlist and database files
<code>run_dir</code>	Run directory

Traditional Synthesis Run

For comparison purposes, you will try a traditional synthesis run of the FIR filter. On the surface, the only difference between the traditional synthesis run and the BGX run is that you invoke a different shell. The two runs use the same design and the same constraints.

The script for the baseline run is in the `tcl` directory and is called `run_no_dp.tcl`. The only difference between this script and the equivalent datapath script is the output filenames. One of the strengths of this solution is that you do not need to make any changes to your scripts or code to take advantage of datapath.

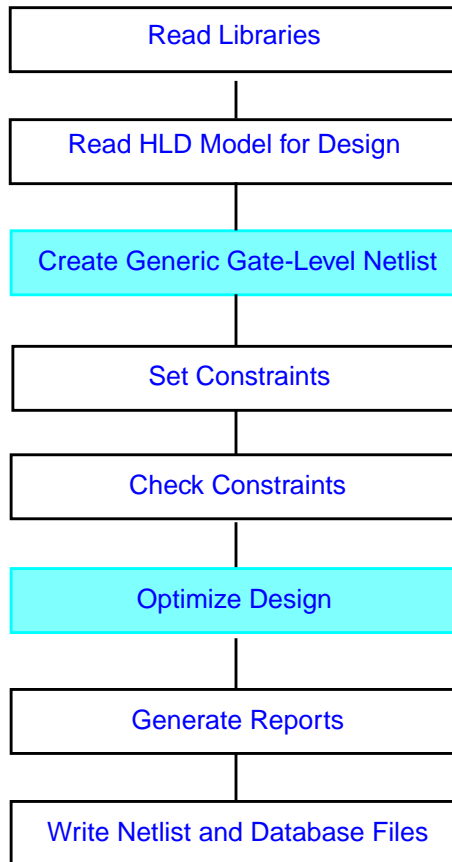
1. Change to the `run_dir` directory in the `tutorial_buildgates` directory.
2. Start the traditional synthesis run by entering the following in a UNIX shell:

```
unix_shell>bg_shell -gui -f ../tcl/run_no_dp.tcl -logfile no_dp.log  
-cmdfile no_dp.cmd &
```

Important

Make sure you run **bg_shell** and not `bgx_shell` at this stage. The goal is to see a non-datapath run. This synthesis run will take a relatively long time to finish depending on the speed of your machine. You can let it run in the background and move on to the BGX run.

BGX Datapath Flow



Note: Colored steps are affected by Datapath.

Cadence Synthesis Rapid Adoption Kit Getting Started with Datapath

Setting up the Design Environment

1. In the `run_dir` directory of the `tutorial_dp` directory, start BuildGates Extreme by entering the following:

```
unix_shell>bgx_shell -gui &
```

2. Set up the design environment by entering the following command in the console:

```
bgx_shell>source ../tcl/setup.tcl
```

The `setup.tcl` file defines all the path variables. If you prefer, you can enter these commands manually in the console.

The content to of the `setup.tcl` file is as follows:

```
set rak_dir ..
set tcl_dir $rak_dir/tcl
set rtl_dir $rak_dir/rtl
set lib_dir $rak_dir/lib
set rep_dir $rak_dir/rpt
set adb_dir $rak_dir/adb
```

Setting up Global Variables

- Set up the design globals:

```
bgx_shell>source $tcl_dir/set_global.tcl
```

The `set_global.tcl` script has the following content:

```
set_global aware_adder_architecture ripple
```

This command sets the initial adder architecture to a ripple adder (see the [*Command Reference for BuildGates Synthesis and Cadence PKS*](#) for details about this global). Based on the design and timing constraints, the final adder architecture might be different. Even for the same design, different initial adder selection may end up with different final architecture. For this design, you will start with a slow but small adder for demonstration purpose. By default BGX starts with FCLA (fast carry look-ahead adder).

Reading the Synthesis Technology Libraries

1. Read the synthesis technology libraries:

```
bgx_shell>source $tcl_dir/read_lib.tcl
```

This command sets the technology libraries and wireload model for this design.

2. View the contents of the library:

```
bgx_shell>report_library
```

The `report_library` command can report on the cells in the library, the wireload models, the operating conditions, and more.

Reading the HDL Models for the Design

```
bgx_shell>source $tcl_dir/read_design.tcl
```

The content of the script is as follows:

```
read_verilog $rtl/filt4.v
```

Creating a Generic Gate-Level Netlist

- Generate the generic build of the netlist:

```
bgx_shell>do_build_generic
```

The `do_build_generic` command transforms the design into a hierarchical, gate-level netlist. The Logical browser and console change as the tool builds the netlist.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Datapath

Setting Constraints

1. Constrain the design:

```
bgx_shell>source $tcl_dir/constraints.tcl
```

This command sets up the ideal clock, binds the clock root to the clock pin, and sets input and external delay for the primary input and output ports.

The `constraints.tcl` file contains the following commands:

```
set_current_module filt4
set_top_timing_module filt4

set cyc 10.00
set_clock CLK -period $cyc -waveform "0 [expr $cyc / 2]"
set_clock_root -clock CLK [find -port SYS_CLK]
set_clock_insertion_delay 0.3 -pin {SYS_CLK}

proc all_inputs {} {find -port -inputs -no_clocks *}
proc all_outputs {} {find -port -outputs *}
set_input_delay -clock CLK 0.4 [all_inputs]
set_external_delay -clock CLK 0.5 [all_outputs]

set_drive_cell -cell DFFX4 -from_pin CP -pin Q [all_inputs]
set_drive_cell -cell CLKBUFX20 SYS_CLK
set_port_capacitance [expr [get_cell_pin_load -cell DFFX4 -pin D]*4.0] \
[all_outputs]

set_false_path -from [find -port -input reset]
```

See the [*Command Reference for BuildGates Synthesis and Cadence PKS*](#) for more details on these commands.

The last command tells the tool not to spend time optimizing any timing path starting from the input signal `reset`.

Note: The clock frequency is 10ns, which means that if timing is met, the design will run at 100 MHz.

2. After applying the timing constraints, check the constraints:

```
bgx_shell>check_timing -verbose
```

The `check_timing` command performs a variety of consistency and completeness checks on the timing constraints specified for a design. As you can see, there is no warning.

3. At this point, check the netlist to make sure there are no problems:

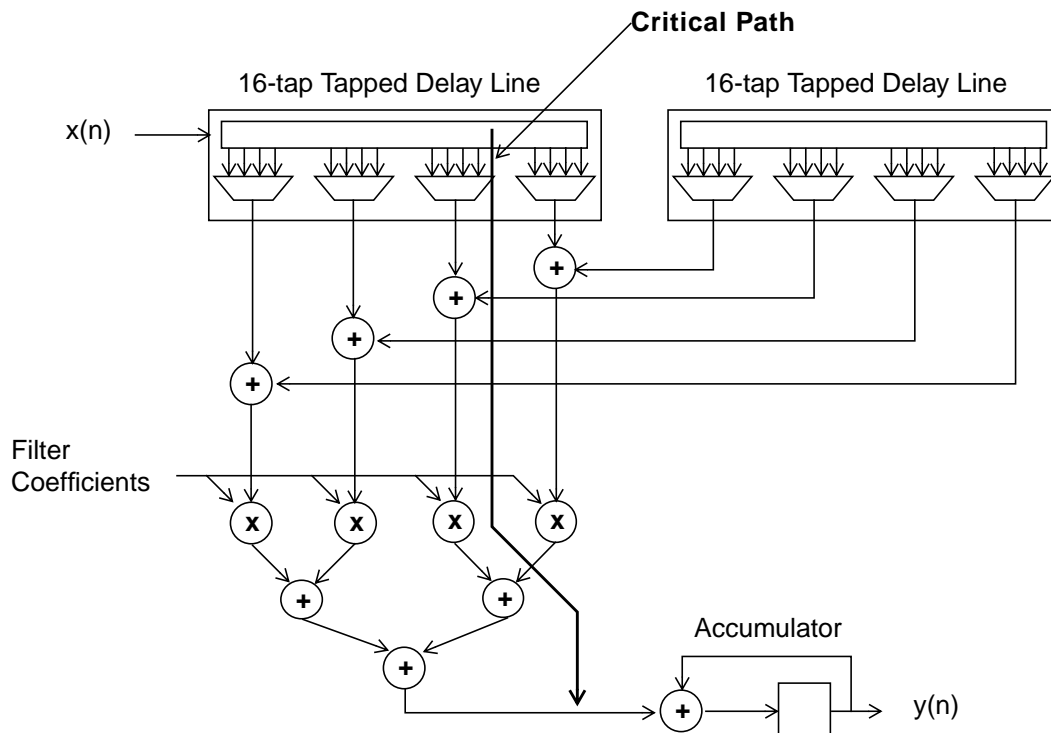
```
bgx_shell>check_netlist -verbose
```

The `check_netlist` command performs a number of checks on the structural connectivity of the netlist. All checks come up clean.

Operator Merging

One of the techniques that BGX uses to improve the speed and decrease the area of a design is operator merging. For more details on this technique, see [More about Operator Merging](#). Figure 4-2 shows how operator merging is applied to the FIR design used in this tutorial.

Figure 4-2 Critical Path for FIR Filter



You can see eight adders and four multipliers in the diagram. There is also one more adder (not shown) in the block right before the final output $y(n)$. The critical path in this design goes from the input through the datapath and to the input of the accumulator, including three adders and one multiplier. The operator-merged result combines the four multipliers and three downstream adders into one big component. You may wonder why all the four upstream adders were not merged into that big component as well. Technically, it is feasible, but the tool chooses not to do so because this would lead to a big area penalty with little timing benefit.

To see the pre-optimized merged component, follow these steps:

1. In the Logical browser, expand the logical hierarchy of module `filt4` by clicking on the + icon left of `filt4`.
2. Double click on module `AWDP_partition_2` using the left mouse button.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Datapath

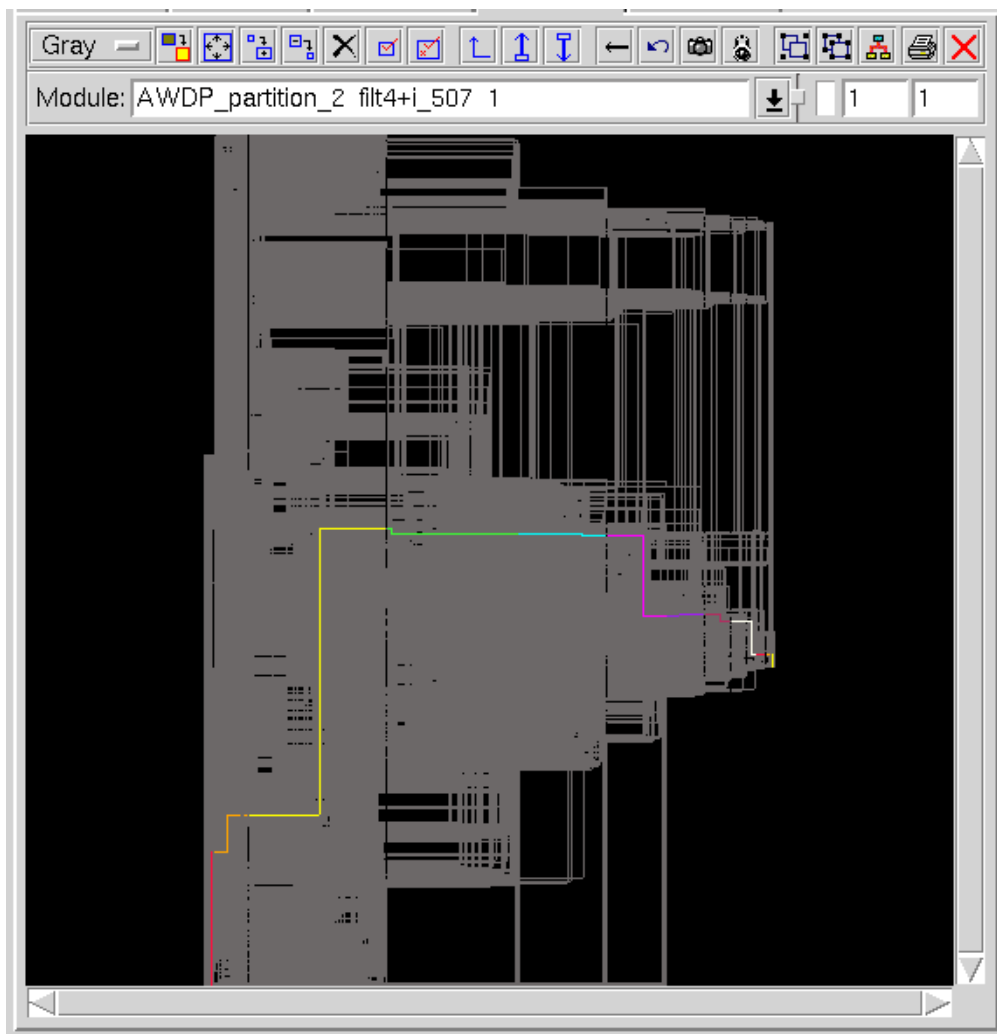
3. Highlight the worst path:

- a. Bring up the pop-up menu by clicking the right mouse button in your Schematic window.
- b. Select *Worst Path*.

The result shown in [Figure 4-3](#) on page 109 shows the pre-optimized version of the merged operators. It contains four multipliers and three adders in one module.

Note: If you cannot reproduce this figure, select *Edit – Preferences – Schematic*, then click the *Paging* tab and set *Page Size* to None. Then double click on module `AWDP_partition_2` (in the Logical browser) to refresh the schematics. Then reselect *Worst Path*.

Figure 4-3 Pre-optimized Critical Path in Merged Component



Cadence Synthesis Rapid Adoption Kit Getting Started with Datapath

To see which architectures are embedded in this merged operator:

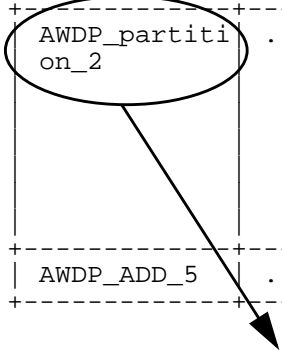
1. Doubleclick the `filt4` module in the Logical browser (this will set `filt4` as the current module).
2. Report the datapath resources in the design with the `report_resources` command:

```
bgx_shell> report_resources -hier
```

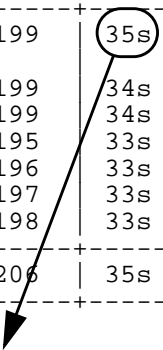
Your report will be similar to the report shown in [Figure 4-4](#) on page 110.

Figure 4-4 Resource Report

Arithmetic Resources							
Module	File	C	Arch	Op	Line	Out	In
AWDP_EQ_1	../rtl/filt4.v	1	cla	==	10	1u	1ux1u
AWDP_EQ_0	../rtl/filt4.v	1	cla	==	16	1u	1ux1u
AWDP_EQ_10	../rtl/filt4.v	1	cla	==	31	1u	1ux1u
AWDP_EQ_00	../rtl/filt4.v	1	cla	==	37	1u	1ux1u
AWDP_ADD_0	../rtl/filt4.v	1	ripple	+	55	16u	16ux1u
AWDP_ADD_3	../rtl/filt4.v	1	ripple	+	191	17s	16sx16s
AWDP_ADD_4	../rtl/filt4.v	1	ripple	+	192	17s	16sx16s
AWDP_ADD_00	../rtl/filt4.v	1	ripple	+	193	17s	16sx16s
AWDP_ADD_1	../rtl/filt4.v	1	ripple	+	194	17s	16sx16s
AWDP_partition_2	../rtl/filt4.v	1	ripple	+	199	35s	34sx34s
				+	199	34s	33sx33s
				+	199	34s	33sx33s
				*	195	33s	17sx16s
				*	196	33s	17sx16s
*	197	33s	17sx16s				
*	198	33s	17sx16s				
AWDP_ADD_5	../rtl/filt4.v	1	ripple	+	205	35s	35sx35s



partition



bit width and sign

This report shows the architectures, the bit widths of the datapath functions, the name of the partition they are in, the magnitude of the component, and whether it is signed or unsigned.

Optimizing the Design

- Optimize the design using the `do_optimize` command:

```
bgx_shell>do_optimize
```

The optimization can take a while to finish depending on the speed of your machine. When complete, you will compare it to the traditional synthesis run to show the QOR improvement.

Generating Reports

When the BGX run finishes, you can examine the results.

1. Report the area and speed of the design:

```
bgx_shell>set_current_module filt4  
bgx_shell>report_timing  
bgx_shell>get_area
```

See the *[Command Reference for BuildGates Synthesis and Cadence PKS](#)* for more information on these commands.

[Figure 4-5](#) on page 112 shows an example of the timing report.

From these reports, it is evident that the design has the following specifications:

- Speed: 100 MHz
- Area: 185423.52 μm^2
- Run Time: approximately 30 minutes (depending on the speed of your machine)

Cadence Synthesis Rapid Adoption Kit Getting Started with Datapath

Figure 4-5 Timing Report after Optimization

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	cycle[1] ^			0.47	0.47
i_177	A ^ -> Y ^	BUFEX12	0.13	0.60	0.60
i_166	B ^ -> Y v	NOR2BX4	0.07	0.67	0.67
i_5548	A v -> Y v	BUFEX20	0.14	0.81	0.81
i_4914	A v -> Y v	BUFEX3	0.14	0.95	0.95
i_837	A1 v -> Y ^	AOI22XL	0.28	1.24	1.24
i_85	A ^ -> Y v	NAND2X2	0.15	1.39	1.39
i_510	data_H1[1] v	AWDP_ADD_4		1.39	1.39
i_510/i_3	A v -> Y ^	XNOR2X4	0.29	1.68	1.68
i_510/i_37	A0 ^ -> Y v	OAI21X4	0.08	1.76	1.76
i_510/i_5842	A v -> Y v	BUFEX2	0.20	1.96	1.95
i_510/i_53	A0 v -> Y ^	AOI21X4	0.20	2.15	2.15
i_510/i_112	A ^ -> Y ^	XOR2X1	0.40	2.55	2.55
i_510	s1[4] ^	AWDP_ADD_4		2.55	2.55
i_507	s1[4] ^	AWDP_partition_2		2.55	2.55
i_507/i_317	A ^ -> Y v	INVX1	0.16	2.71	2.71
i_507/i_4595	A v -> Y v	BUFEX8	0.19	2.90	2.90
i_507/i_399	B v -> Y ^	NOR2X1	0.19	3.09	3.09
i_507/i_1638	B ^ -> CO ^	ADDFX1	0.57	3.66	3.66
i_507/i_1698	A ^ -> CO ^	AFHCINX2	0.45	4.11	4.11
i_507/i_3530	B ^ -> Y ^	XOR2X2	0.31	4.43	4.43
i_507/i_3531	A ^ -> Y ^	XOR2X1	0.29	4.72	4.72
i_507/i_1776	CI ^ -> CO ^	ADDFX2	0.30	5.03	5.03
i_507/i_1851	A ^ -> S ^	AFHCINX2	0.50	5.53	5.53
i_507/i_1861	B ^ -> CO ^	ADDFHX1	0.38	5.91	5.91
i_507/i_1924	A ^ -> S v	ADDFHX4	0.49	6.40	6.39
i_507/i_1928	B v -> CON ^	AFHCINX2	0.34	6.74	6.74
i_507/i_1985	CIN ^ -> S v	AFHCINX2	0.34	7.08	7.08
i_507/i_1987	A v -> CO v	AFHCINX2	0.39	7.47	7.47
i_507/i_2038	CI v -> CO v	ADDFHX2	0.25	7.72	7.72
i_507/i_2085	A v -> S v	AFHCINX2	0.52	8.24	8.24
i_507/i_2344	A v -> Y ^	NOR2X2	0.19	8.43	8.42
i_507/i_2521	A ^ -> Y v	NOR2XL	0.13	8.55	8.55
i_507/i_1388	A v -> Y v	BUFEX4	0.17	8.72	8.72
i_507/i_2555	A1 v -> Y ^	AOI21X1	0.26	8.98	8.98
i_507/i_2624	B0 ^ -> Y v	OAI21X1	0.21	9.20	9.19
i_507/i_2701	A0 v -> Y ^	AOI21X2	0.18	9.38	9.38
i_507/i_2770	B0 ^ -> Y v	OAI21X2	0.11	9.48	9.48
i_507/i_2839	B0 v -> Y ^	AOI21X2	0.15	9.63	9.63
i_507/i_3799	S0 ^ -> Y v	MXI2X1	0.20	9.83	9.83
i_507	mult_out[32] v	AWDP_partition_2		9.83	9.83
u3	d[32] v	reg_35_0		9.83	9.83
u3/q_reg_32	D v	SDFFRHQXL	0.00	9.83	9.83

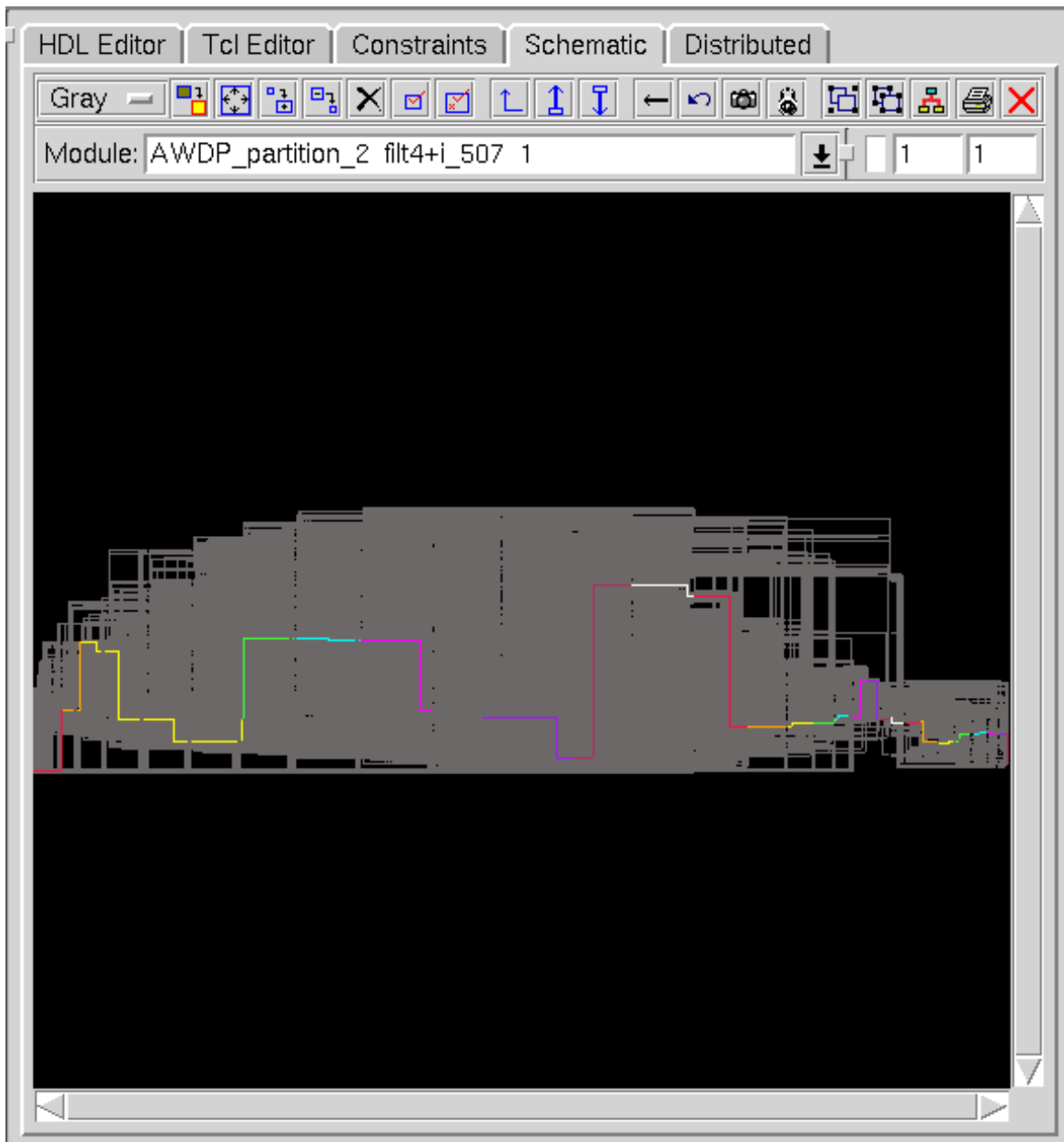
2. In the Logical browser double click on module AWDP_partition_2 using the left mouse button.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Datapath

3. Highlight the worst path:
 - a. Bring up the pop-up menu clicking and holding the right mouse button in your Schematic window.
 - b. Select *Worst Path*.

The result is shown below:



Cadence Synthesis Rapid Adoption Kit Getting Started with Datapath

Note: The automatic architecture selection chose a faster adder to meet timing. (See also [More about Automatic Architecture Selection](#).)

4. Check which adder was used:

```
bgx_shell> set_current_module filt4
bgx_shell> report_resources -hier
```

Figure 4-6 on page 114 shows the resulting resource report.

The tool upsized the adders from RIPPLE to CLA or FCLA on the critical path in AWDP_partition_2.

Figure 4-6 Resource Report After Optimization

Arithmetic Resources							
Module	File	C	Arch	Op	Line	Out	In
AWDP_ADD_0	../rtl/filt4.v	1	ripple	+	55	16u	16ux1u
AWDP_ADD_3	../rtl/filt4.v	1	cla	+	191	17s	16sx16s
AWDP_ADD_4	../rtl/filt4.v	1	cla	+	192	17s	16sx16s
AWDP_ADD_00	../rtl/filt4.v	1	cla	+	193	17s	16sx16s
AWDP_ADD_1	../rtl/filt4.v	1	cla	+	194	17s	16sx16s
AWDP_partiti on_2	../rtl/filt4.v	1	fcla	+	199	35s	34sx34s
				+	199	34s	33sx33s
				+	199	34s	33sx33s
				*	195	33s	17sx16s
				*	196	33s	17sx16s
				*	197	33s	17sx16s
			booth	*	198	33s	17sx16s
AWDP_ADD_5	../rtl/filt4.v	1	ripple	+	206	35s	35sx35s

Writing Netlist, Database Files and Report File

1. Write out the Verilog netlist and database.

```
bg_shell> write_verilog -hierarchical $adb_dir/filter_dp.v
bg_shell> write_adb $adb_dir/filter_dp.adb
```

2. Write out the results to a report file:

```
report_timing > $rep_dir/filter_dp.rpt
report_area >> $rep_dir/filter_dp.rpt
report_resources -hier >> $rep_dir/filter_dp.rpt
```

3. Select *File – Exit* or type `exit` in the console to end this BGX session.

Examining Traditional Synthesis Results

To examine the results of traditional synthesis, turn to the `bg_shell` run you started at the beginning of the tutorial. If it is still running, you either let it go to completion, or you can interrupt it by clicking on the *Stop* icon under menu bar.

1. To get the area and speed of the design, enter the following:

```
bg_shell> report_timing
bg_shell> get_area
```

From these reports, it is evident that the design has the following specifications:

- ❑ Speed: 94 MHz
- ❑ Area: 230393.12 μm^2
- ❑ Run Time: approximately 90 minutes (depending on the speed of your machine)

Note: The reported slack is -0.69 ns. Since the design was compiled for 100 MHz, this translates into a maximum frequency of approximately 94 MHz.

2. Select *File – Exit* or type `exit` in the console to end the traditional synthesis session.

Comparing BGX and Traditional Results

[Figure 4-7](#) on page 115 compares the speed and area of these two results.

Figure 4-7 Comparing Results

	Area (μm^2)	Speed
Normal flow	230393.12	94 MHz
DP	185423.52	100 MHz

Comparing these two results, notice that both tools start with the same design and with the same constraints, and use the same script. By using BuildGates Extreme, you achieved substantial area, timing, and run-time savings.

These results come to you without the difficulty of manual partitioning, without specialized datapath knowledge, and without manual intervention in your synthesis process. After automatically partitioning your design, the BGX tool works to close timing automatically between datapath and control logic, while applying several highly specialized datapath optimizations: automatic architecture selection, automatic operator merging, and Wallace tree optimizations.

Summary

In this session, you

- Learned about BuildGates Extreme (BGX) datapath features.
- Compared the quality of results from BuildGates Extreme with the results from a traditional synthesis run.

Getting Started with Low Power Synthesis

- [Introduction](#) on page 118
- [Objectives](#) on page 118
- [Example Design](#) on page 119
- [Setting Up for This Module](#) on page 119
- [Regular BGX Flow with Power Estimation](#) on page 122
- [BGX LPS Flow](#) on page 123
 - [Setting up Design Environment](#) on page 124
 - [Reading the Synthesis Technology Libraries](#) on page 124
 - [Reading the HDL Models for the Design](#) on page 125
 - [Creating Generic Gate-level Netlist and Exploring Sleep-mode Options at the RTL Level](#) on page 125
 - [Setting Constraints](#) on page 130
 - [Setting Sleep-Mode and Clock-Gating Options](#) on page 130
 - [Optimizing the Design](#) on page 132
 - [Simulating and Reading the Toggle Count File](#) on page 135
 - [Optimizing the Power](#) on page 137
 - [Analyzing the Results and Generating Reports](#) on page 137
 - [Power Analysis in BuildGates Extreme](#) on page 141
 - [Generating Reports](#) on page 148
 - [Writing Netlist and Database Files](#) on page 148
- [Gate-Level Only Flow](#) on page 149
- [LPS-DFT flow](#) on page 149

- [Summary](#) on page 151

Introduction

Today, low power is emerging as a critical issue in ASIC design. The primary driving factors are the remarkable success and growth of personal computing devices (portable desktops, audio- and video-based multimedia products) and wireless communications systems (personal digital assistants, pagers, and cellular phones). These technologies demand high-speed computation and complex functionalities, which means low-power consumption is a crucial design concern.

Another development that emphasizes the need for low-power solutions is the growing number of large, high-performance designs. Millions of transistors switching at high clock speeds lead to power consumption problems that can be difficult to solve late in the design cycle. The high cost associated with packaging and cooling strategies is yet another factor that makes it imperative to have a seamless low-power methodology.

Because power dissipation is becoming just as important as performance and area, power analysis and optimization needs to be an integral part of the design methodology. The Cadence approach with Low-Power Synthesis (LPS) is to address low-power issues early in the design cycle and fix them automatically during the optimization flow. LPS explores options for lowering power at the RTL level, but does not make any commitments until the gate level. Also, all gate-level transformations take into account power, delay, and placement information. This option lets engineers determine a design's power dissipation early in the design process which is a key benefit for chip designers targeting power-sensitive applications.

Objectives

In this module, you will run the regular BGX flow and the BGX-LPS flow in parallel on the same design. You will learn more about the

- LPS features while running the BGX-LPS flow
- LPS automatic power saving capability when comparing the results of both flows

At the end of this module, the gate-level only LPS flow and LPS DFT flow are briefly addressed.

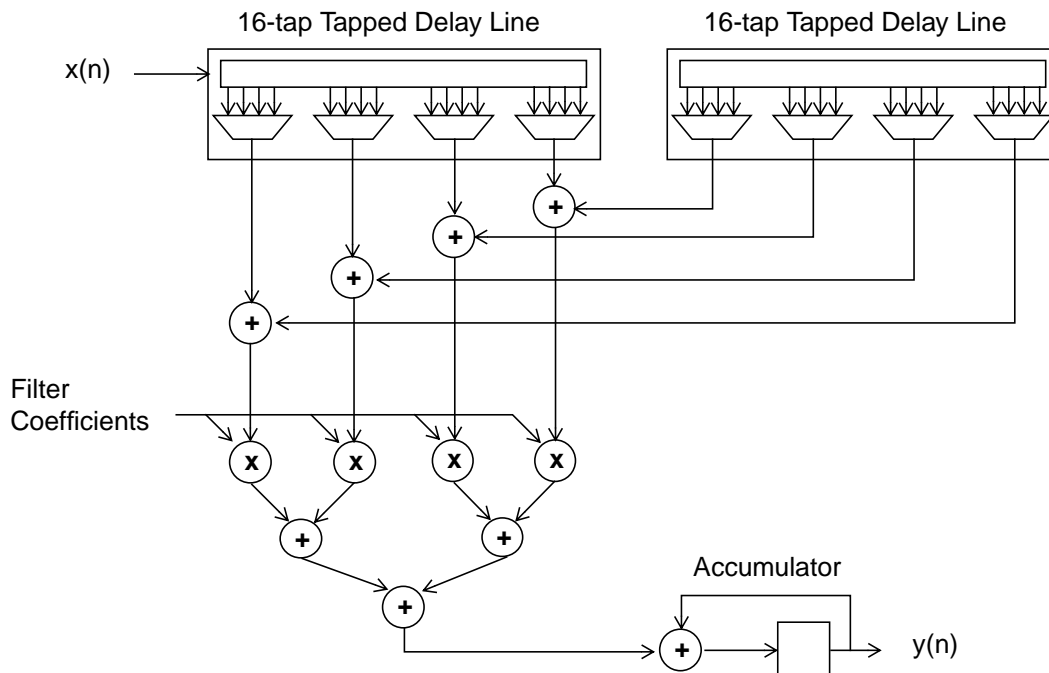
Note: This module assumes that you completed [Chapter 2, “Getting Started with BuildGates Synthesis.”](#)

Example Design

You will use an FIR filter (shown in Figure 5-1) with the following specifications:

- 32-Tap
- 16-bit coefficient
- 16-bit I/O data

Figure 5-1 FIR Filter Architecture



Setting Up for This Module

Note: We assume that you already copied the data. If not, see [Getting Started](#) on page 12.

Required License

You need a BuildGates Extreme (BGX) license to run this module. To save running time you can run two copies of `bgx_shell` in parallel, one for the regular flow and one for the LPS flow. This implies that you need to check out two BGX licenses. You can still complete the whole flow with only one license, but it will take longer as you need to run two flows one after the other. Call your sales representative to get demo licenses if needed.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

Ensure Access to a Verilog Simulator

To do any meaningful power optimization, you need to simulate the design. LPS supports all simulators. You can either use a Cadence[®] simulator (Cadence[®] NC-Verilog[®] simulator, or Verilog[®]-XL simulator) or another vendor's Verilog simulator. In this tutorial, we use the Verilog-XL simulator.

Note: If you want to use the Verilog-XL simulator, you need to use version 2.7 or higher.

Locate the path to your simulator, type the following command in a UNIX window:

```
which verilog
```

If “no verilog” is returned, contact your system administrator to give you the installation directory to the Verilog simulator and set up the environment accordingly.

Link to the Simulation Library

While running the simulation you need to link to the LPS PLI shared library to capture the design switching activity during the simulation.

Note: You can skip this step if you call the simulator within the `bgx_shell` environment.

■ SunOS/Solaris:

```
setenv LD_LIBRARY_PATH  
your_install_path/version/lib/archives/sunos7/sparc:$LD_LIBRARY_PATH
```

■ HPUX:

```
setenv SHLIB_PATH  
your_install_path/version/lib/archives/hpux11/hppa32:$SHLIB_PATH
```

■ IBM:

```
setenv LD_LIBRARY_PATH  
your_install_path/version/lib/archives/aix43/ppc32:$LD_LIBRARY_PATH
```

Required Files

You need the following files when you start the BuildGates-LPS flow:

■ Simulation library

A complete set of Verilog files for the library cells.

■ Synthesis library

A `.t1f` or `.a1f` file containing either cell-based or arc-based power tables. BGX supports both the traditional 2-D look-up table and the 3-D look-up table. In case of the 3-D table, the internal power is not only a function of the input slew and output load capacitance, but also of the sum of the capacitances on the other output pins.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

- Verilog or VHDL netlist, preferably at RTL level.
- Testbench

Unlike your traditional testbench, which is designed for coverage, you need a testbench that simulates how the design works in real life. To emphasize power savings, consider exercising the parts of the design that are used most during power sensitive application. For example, if you were designing a digital camera, consider emphasizing the photo taking process and de-emphasizing the save to hard drive process.

- Constraints

The *your_install_path/version/tutorial/RAK/tutorial_lps* directory contains the files needed to run this module and has the following directory structure:

<code>tutorial_lps</code>	Root directory
<code>tcl</code>	Scripts to run the RAK LPS module
<code>rtl</code>	Verilog design modules
<code>lib</code>	Synthesis libraries and simulation libraries
<code>rpt</code>	Design outputs including timing, area, hierarchy and other report files
<code>adb</code>	Design outputs including netlist and database files
<code>simulation</code>	Simulating files and test benches
<code>run_dir</code>	Running directory

Regular BGX Flow with Power Estimation

You will start a traditional synthesis run with power estimation. Later, you will compare its results with those of the LPS flow to see the automatic power saving capability of LPS. A regular BGX flow with power estimation contains these steps:

```
read_alf                                # or read_tlf
read_verilog                             # or read_vhdl
do_build_generic
# Set timing constraints
do_optimize write_verilog -hier
# Simulate                               # generates toggle count format (TCF) file
# Write netlist and database files
# Generate reports and analyze the design
```

The TCF file recording the switching activities of the circuit will be used by LPS to estimate the power consumption of the design. For more information about TCF file, refer to [Simulating and Reading the Toggle Count File](#) on page 135.

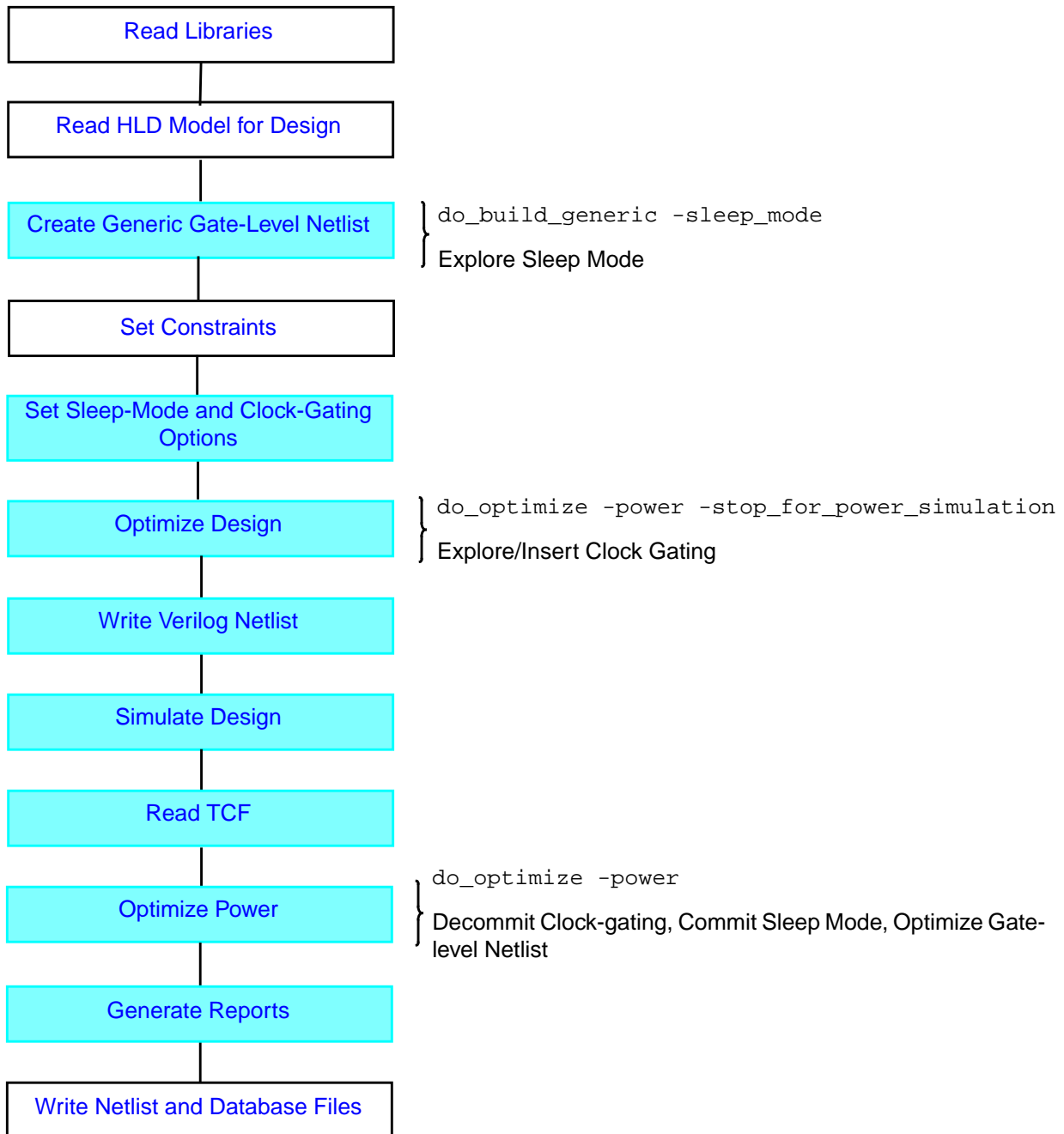
To execute this flow, follow these steps:

1. Change the current directory to `tutorial_lps/run_dir` in the directory you created.
2. Start BGX by entering the following command:

```
unix_shell>bgx_shell -gui -f ../tcl/run_no_lps.tcl -logfile run_no_lps.log
-cmdfile run_no_lps.cmd &
```

While this process runs in the background, you can start the BGX LPS flow. The sample script, `run_no_lps.tcl`, is provided in the `tcl` directory.

BGX LPS Flow



Note: Colored steps are either LPS-specific or have specific LPS options.

The following sections explain these steps in detail. The Tcl script for this flow, `run_lps.tcl`, is provided in the `tcl` directory.

Setting up Design Environment

1. From the `run_dir` directory, invoke BuildGates Extreme in graphical user interface (GUI) mode:

```
unix_shell>bgx_shell -gui &
```

2. Set up your design environment by running the following script in the console of the BuildGates Synthesis window:

```
bgx_shell>source ../tcl/setup.tcl
```

The `setup.tcl` file defines all path variables and has the following content:

```
set rak_dir      ..
set tcl_dir      $rak_dir/tcl
set rtl_dir      $rak_dir/rtl
set lib_dir      $rak_dir/lib
set rep_dir      $rak_dir/rpt
set adb_dir      $rak_dir/adb
set sim_dir      $rak_dir/simulation
```

Reading the Synthesis Technology Libraries

1. Read in the synthesis technology libraries by running the following script:

```
bgx_shell>source $tcl_dir/read_lib.tcl
```

The `read_lib.tcl` script contains the following statements:

```
read_tlf $lib_dir/slow_4.3.tlf
read_symbol $lib_dir/slow.sym
```

2. View the contents of the technology library with the following command:

```
bgx_shell>report_library
```

The `report_library` command reports on the cells in the library, the wireload models, the operating conditions, and more. Since you only read one library, you do not need to specify the library name here.

3. Ensure that the design does not map to cells without power characterization with the following command:

```
bgx_shell>check_library -power library_pwr.rpt
```

The `check_library` command generates a file containing all the cells without power characterization.

4. Prevent the design from mapping into those cells with the following command:

```
bgx_shell>foreach i [exec cat library_pwr.rpt] \
  {set_cell_property dont_utilize true $i}
```

This command sets some cell properties on the cells without power characterization.

Or you can run the following script which executes steps 3 and 4:

```
bgx_shell>source $tcl_dir/check_lib.tcl
```

Reading the HDL Models for the Design

- After loading the libraries, read the RTL code for this design:

```
bgx_shell>source $tcl_dir/read_design.tcl
```

The `read_design.tcl` script has the following content:

```
read_verilog $rtl_dir/filt4.v
```

Creating Generic Gate-level Netlist and Exploring Sleep-mode Options at the RTL Level

1. Transform the RTL netlist you just read into a hierarchical, gate-level netlist consisting of technology- independent logic gates.

```
bgx_shell>do_build_generic -sleep_mode
```

The `-sleep_mode` option of the `do_build_generic` command tells LPS to explore areas of the design where power could be saved using the sleep-mode technique.

Note: The sleep-mode logic is only committed during gate-level power optimization based on timing, area and power.

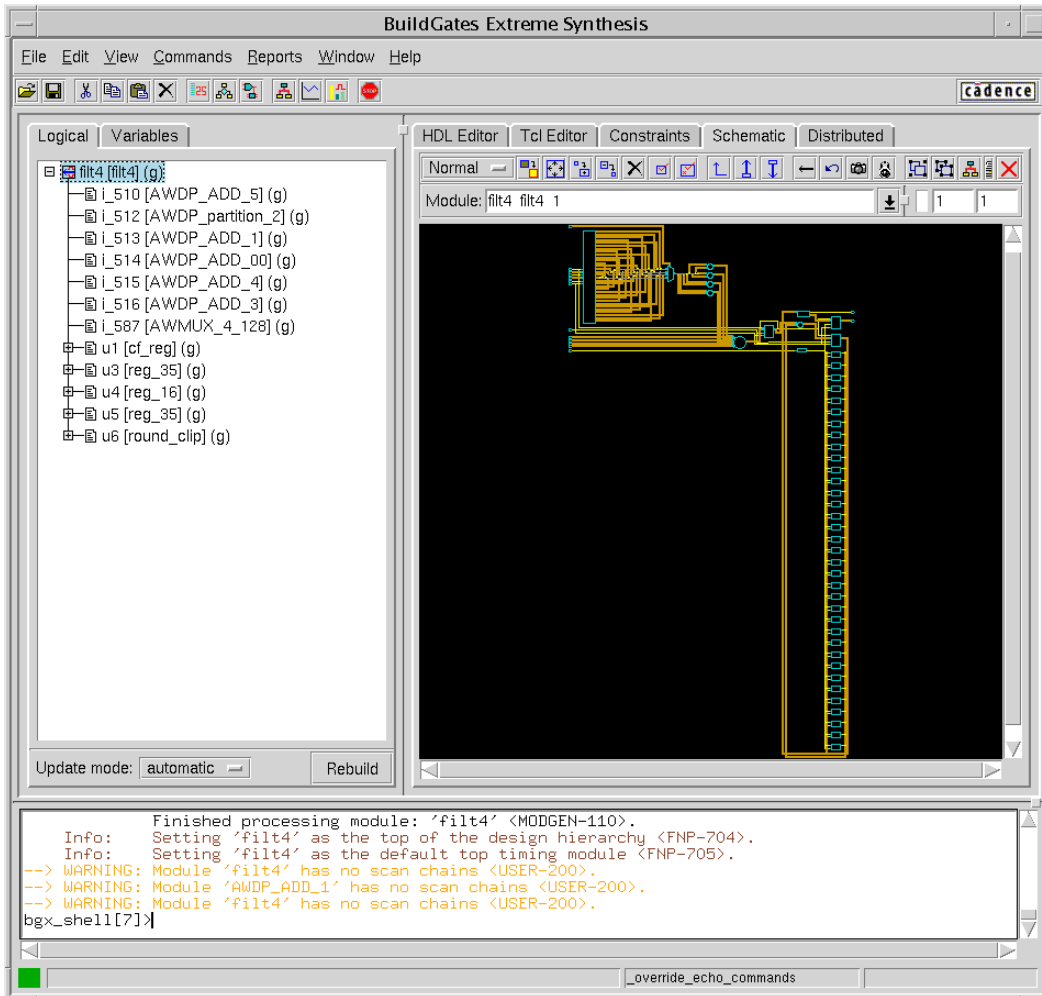
2. Using the left mouse, click on the + icon next to `filt4` to see the hierarchy beneath.
3. Display the schematic generated by `do_build_generic` by double clicking on top module `filt4` in the Logical browser using the left mouse button.

The schematic viewer window is activated. The schematic is displayed (See Figure [5-2](#)).

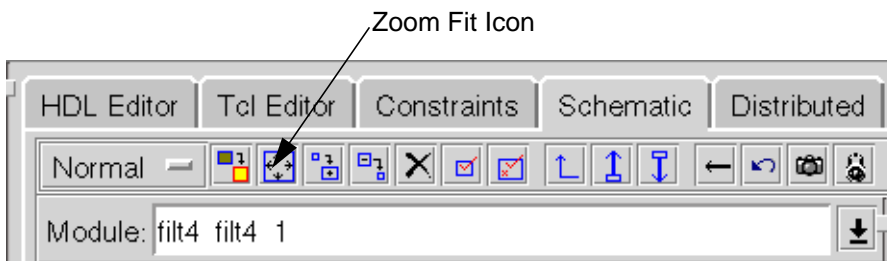
Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

Figure 5-2 Schematic of Top Module filt4



Note: If you get a zoomed out view of the schematic, use the Zoom Fit icon on the tool bar under the Schematic tab to fit the schematic in the Schematic window.



4. Select *Reports – Sleep Mode Logic Info* to list the functional blocks that were marked as sleep-mode candidates.

The Sleep Mode Information window is displayed.

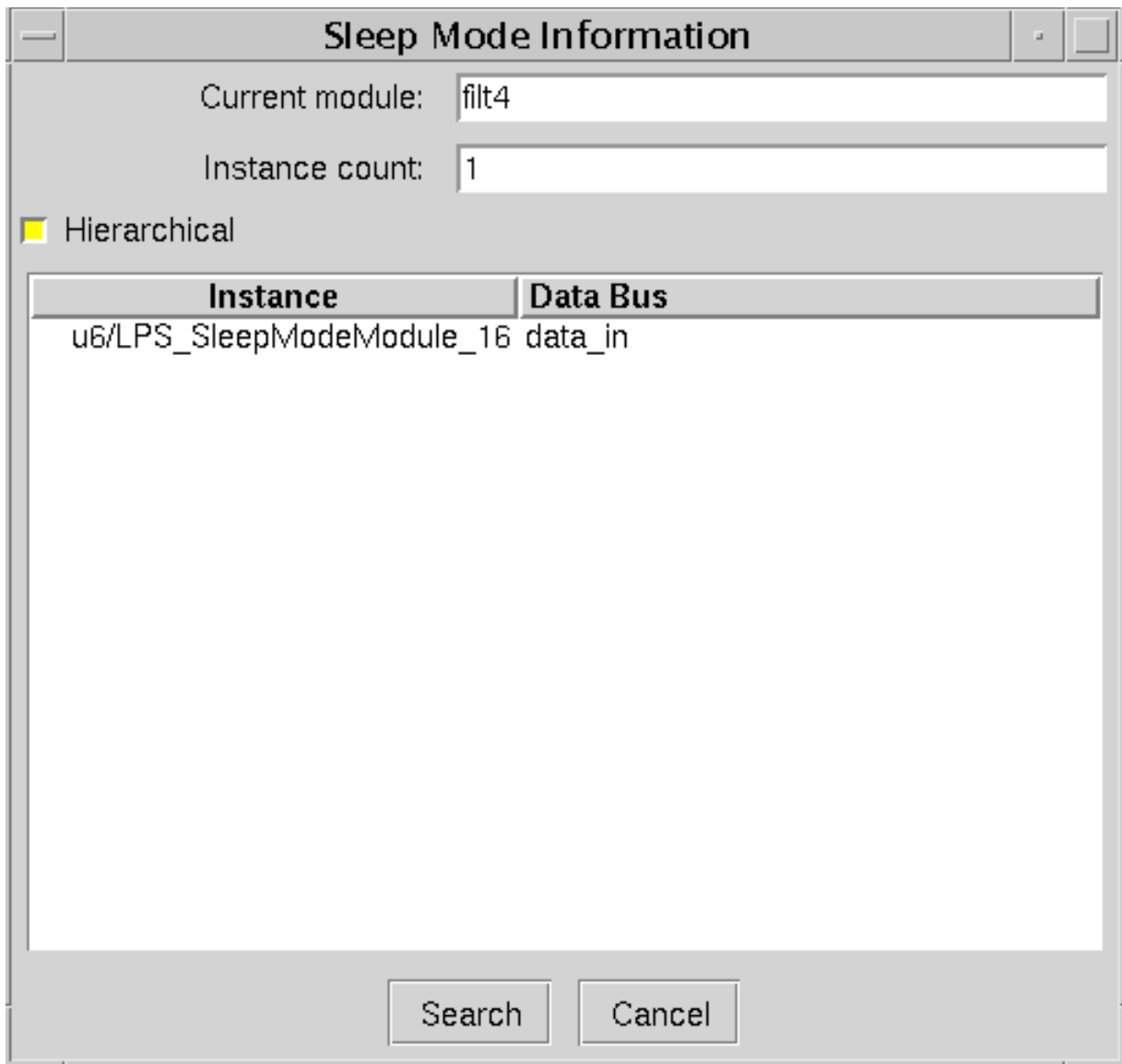
Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

- a. Select *Hierarchical*.
- b. Click the *Search* button to populate the window.

The result is shown in [Figure 5-3](#) on page 127. One sleep-mode candidate LPS_SleepModeModule_16 was found.

Figure 5-3 Sleep Mode Information Window



- c. Click the left mouse button on the sleep-mode candidate in the Sleep Mode Information window.

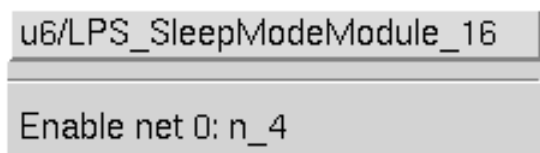
The sleep mode module is highlighted in the schematic.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

- d. Click the right mouse button on the sleep-mode candidate in the Sleep Mode Information window.

A window with the components of the sleep-mode logic pops up.



- e. Click *Cancel* to close the Sleep Mode Information window.
5. List all instance IDs of candidates for sleep-mode information in the current module.

```
bgx_shell>get_sleep_mode_instance_list -hier
```

The result consists of one triplet, where the first number identifies the sleep-mode module. The second number identifies the data bus. The third number identifies the enable signal.

If the current module is not `filt4` (or `round_clip`), you will not be able to see the `LPS_SleepModeModule_16`. In that case, type `set_current_module filt4` and try again.

Examining RTL Code of the `round_clip` Module

Examining the output in the console, shows that the sleep mode candidate was added to the `round_clip` module. Look at the RTL code for the `round_clip` module which is excerpted from `filt4.v` under the `rtl` directory:

```
module round_clip (clipped, data_out, data_in);
    input [34:0] data_in;           // <35,4,t>
    output [15:0] data_out;        // <16,0,t>
    output clipped;
    wire carry;
    wire [14:0] lower;
    wire [15:0] upper, cap, rnd;
    assign upper = {data_in[34], data_in[29:15]};
    assign lower = {data_in[15], data_in[13:0]};
    assign carry = (| lower) & data_in[14];
    assign rnd = upper + carry;
    assign cap = {data_in[34], {15{~data_in[34]}}};
    assign clipped = data_in[34] ? (~ (& data_in[33:30])) : (| data_in[33:30]);
    assign data_out = clipped ? cap : rnd;
endmodule // round_clip
```

Look at the last assign statement:

```
assign data_out = clipped ? cap : rnd;
```

Cadence Synthesis Rapid Adoption Kit

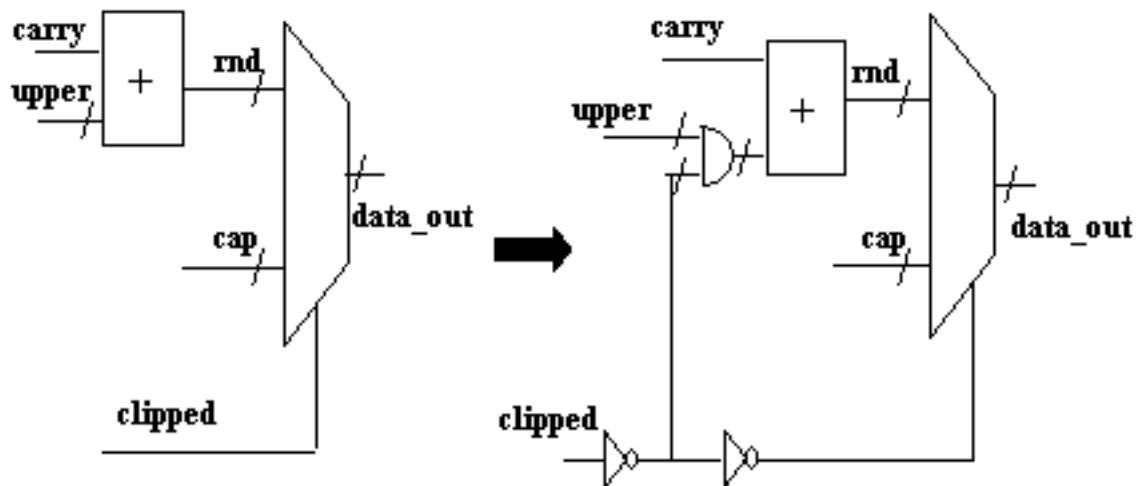
Getting Started with Low Power Synthesis

The output of this module, `data_out`, is controlled by signal `clipped`. If `clipped` is high, `data_out` equals `cap`, otherwise if `clipped` is low, `data_out` equals `rnd`, which is the output of an adder, as indicated by

```
assign rnd = upper + carry;
```

That is a 16 plus 1 bit addition: inputs to the adder are bus `upper` and wire `carry`. Whenever the value of `upper` or `carry` changes, the adder calculates the value for signal `rnd`. However, the signal `rnd` is only needed when the signal `clipped` is low. So LPS sees a power saving opportunity here and inserts AND gates in front of the adder inputs as shown in Figure 5-4. The number of additions could potentially drop depending on how often the input signals toggle and what the probability is that signal `clipped` is high.

Figure 5-4 LPS power saving opportunity



At this point, LPS has only identified this sleep-mode candidate as an opportunity for sleep mode. LPS dynamically computes power savings for each opportunity as well as the cost of implementation—one array of AND gates and two inverters in this example. This dynamic computation is done considering the simulation-based switching information, the capacitance of the nets or gates and the constraints in the design. (You will apply constraints in the next step.) After optimization, only opportunities that actually save power and that do not violate timing paths are kept.

Setting Constraints

1. Set the timing constraints by running the following script.

```
bgx_shell>source $tcl_dir/constraints.tcl
```

The `constraints.tcl` file sets up the ideal clock, binds the clock root to the clock pin, and sets input and external delays for primary input and output ports. This script has the following content:

```
set_current_module filt4
set_top_timing_module filt4

set cyc 10.00
set_clock CLK -period $cyc -waveform "0 [expr $cyc / 2]"
set_clock_root -clock CLK [find -port SYS_CLK]
set_clock_insertion_delay 0.3 -pin {SYS_CLK}

proc all_inputs {} {find -port -inputs -no_clocks *}
proc all_outputs {} {find -port -outputs *}
set_input_delay -clock CLK 0.4 [all_inputs]
set_external_delay -clock CLK 0.5 [all_outputs]

set_drive_cell -cell DFFX4 -from_pin CK -pin Q [all_inputs]
set_drive_cell -cell CLKBUFX4 SYS_CLK
set_port_capacitance [expr [get_cell_pin_load -cell DFFX4 -pin D]*4.0] \
    [all_outputs]
set_false_path -from [find -port -input reset]
```

The last command instructs the tool not to optimize any timing path starting from the input signal `reset`.

2. Check the validity of the constraints.

```
bgx_shell>check_timing -detail
```

No warnings are issued.

3. Check the netlist to make sure there are no problems.

```
bgx_shell>check_netlist -verbose
```

All checks come up clean.

Setting Sleep-Mode and Clock-Gating Options

Next, you will check the sleep-mode and clock-gating options.

1. Check the default values of the sleep-mode options.

```
bgx_shell>get_sleep_mode_options
```

The `get_sleep_mode_options` command returns the table shown in [Figure 5-5](#) on page 131. Only options that are set or for which the tool has defined a default are reported.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

Figure 5-5 Default Sleep-mode Options

Sleep Mode Options	
Options	Value
-no_dissolve	true
-timing_driven	full

By default, LPS does not dissolve the hierarchy of a sleep-mode module when it is committed and it commits each sleep mode module only if it does not violate timing.

2. Check the default values of the clock-gating options.

```
bgx_shell>get_clock_gating_options
```

The `get_clock_gating_options` command returns the table shown in Figure 5-6.

Figure 5-6 Default Clock-Gating Options

Clock-Gating Options	
Options	Value
-auto_test_port	false
-control	none
-control_mode	use_test_mode
-control_port	NOT SET
-domain	all
-force	NOT SET
-gating_style	latch
-ignore	NOT SET
-max_fanout	NOT SET
-minsize	3
-no_timing	false
-observe	false
-obs_style	port
-remove_all	false
-rg_dissolve	false
-rg_force	NOT SET
-rg_ignore	NOT SET
-rg_max	10
-same_polarity	false
-xor_depth	5
-reset	false
-exclude_cells	NOT SET

Most options are related to Design for Test (DFT). For more information on how to set some of these options, refer to [Getting Started with Design For Test](#) on page 73. For this module, you can use default values.

Optimizing the Design

1. Map the design to the target technology and insert clock-gating elements.

```
bgx_shell>do_optimize -power -stop_for_power_simulation
```

The `do_optimize` command maps a generic cell netlist to a technology-specific cell netlist and optimizes the netlist for timing according to the constraints. Logic optimization includes uniquification, constant propagation, structuring, redundancy removal, technology mapping, timing-driven optimization, component swapping, buffering of multi-port nets, and design-rule fixing.

Since you specified `stop_for_power_simulation`, the tool stops post-mapping optimization just before power optimization is performed. Up till then LPS only explores the power saving opportunities. After this point you can run a simulation and read in the toggle count information. Based on the toggle count information LPS can perform all types of gate-level transformations to simultaneously optimize power, delay, and area.

Note: Depending on the performance of your machine, the synthesis can run for a while. You can take this time to read more about the clock-gating techniques used by LPS. Refer to [More about LPS Clock Gating Technique](#) on page 170.

2. Scroll the console back to see what the tool has done so far.

The information shows that BuildGates was working on dissolving instances, duplicating and structuring modules, propagating constants, removing redundancies, exploring clock-gating, inserting clock-gating logic, mapping modules, optimizing modules to meet constraints, fixing design rules, and swapping datapath components.

During clock-gating exploration, LPS identifies all the clock-gating candidates and creates one gating logic per register bank with the same enable signal.

LPS also creates a dummy module that has only one input port connected to the enable signal for the clock gating. LPS needs this dummy module to identify the clock-gating candidates and their corresponding enable functions during decommitment. After gate-level timing optimization, LPS performs decommitment of the clock-gating logic based on timing and power information.

3. Examine one register in the schematic viewer to better understand how clock-gating logic works.

- a. In the Logical browser, first click on the + icon in front of `filt4`, then click on the + icon in front of instance `u1` (module name `cf_reg`) to expand its hierarchy.

You see a list of 32 register banks.

- b. Click on instance `r00` (module name `reg_16_1`) to highlight it.

- c. Bring up the pop-up menu by clicking the right mouse button in the Logical browser.

Cadence Synthesis Rapid Adoption Kit

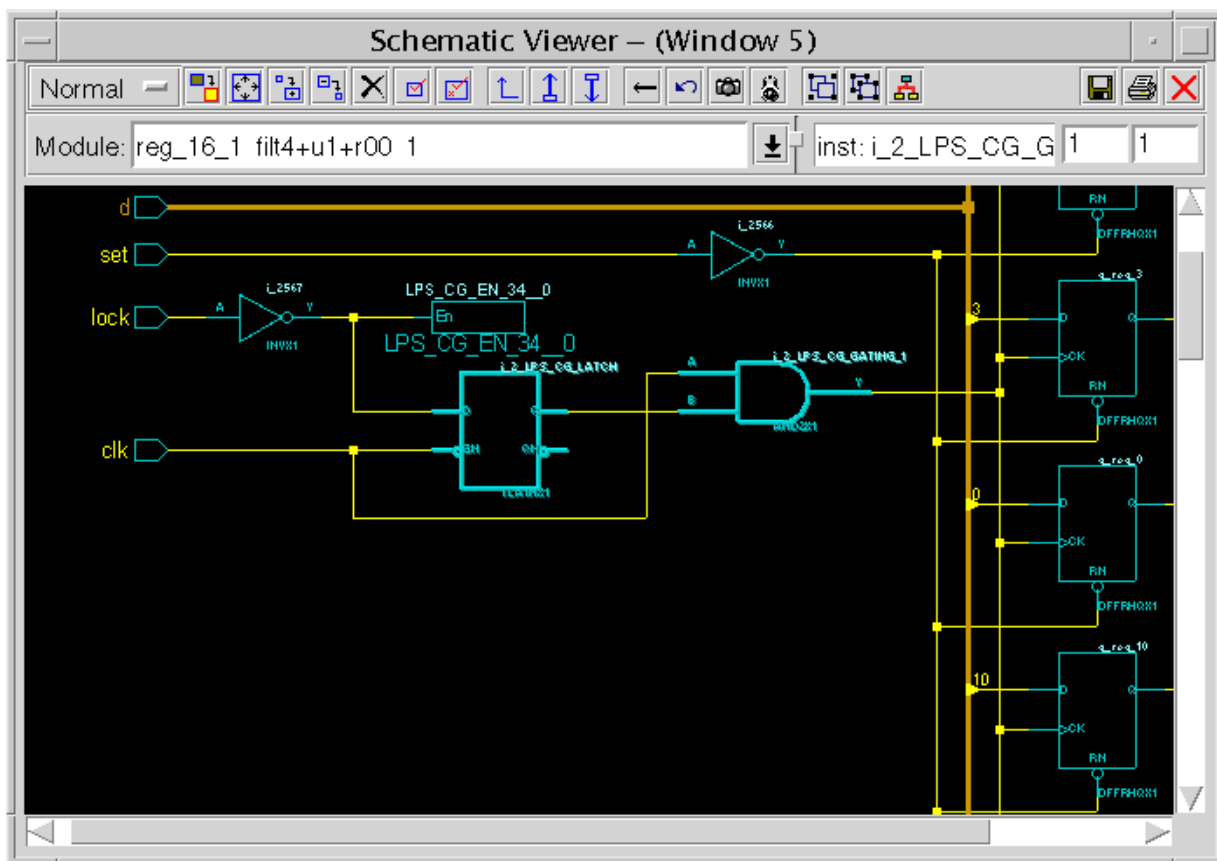
Getting Started with Low Power Synthesis

- d. Select *Open Schematic – New Window* from the pop-up menu and release the right button.

A new Schematic window pops up for the selected instance.

In Figure 5-7, three components were manually highlighted: a latch, a gating cell (AND gate) and a dummy module, LPS_CG_EN_34_0. The dummy module allows LPS to mark the design. You can remove it after the final optimization using the `do remove cg dummy hierarchy` command, but once you issue this command, LPS no longer keeps track of gating instances. Do not issue this command now!

Figure 5-7 Clock-Gating Logic



Note: The output signal from the gating cell drives all 16 registers in this register bank.

- e. Close the Schematic window by clicking on the Close Window icon (red cross).
4. Check how many gating logic circuits exist in this stage.
- a. Double click on `filt4` in the Logical browser to make it the current module.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

- b. Select *Reports – Clock Gating Logic Info*.

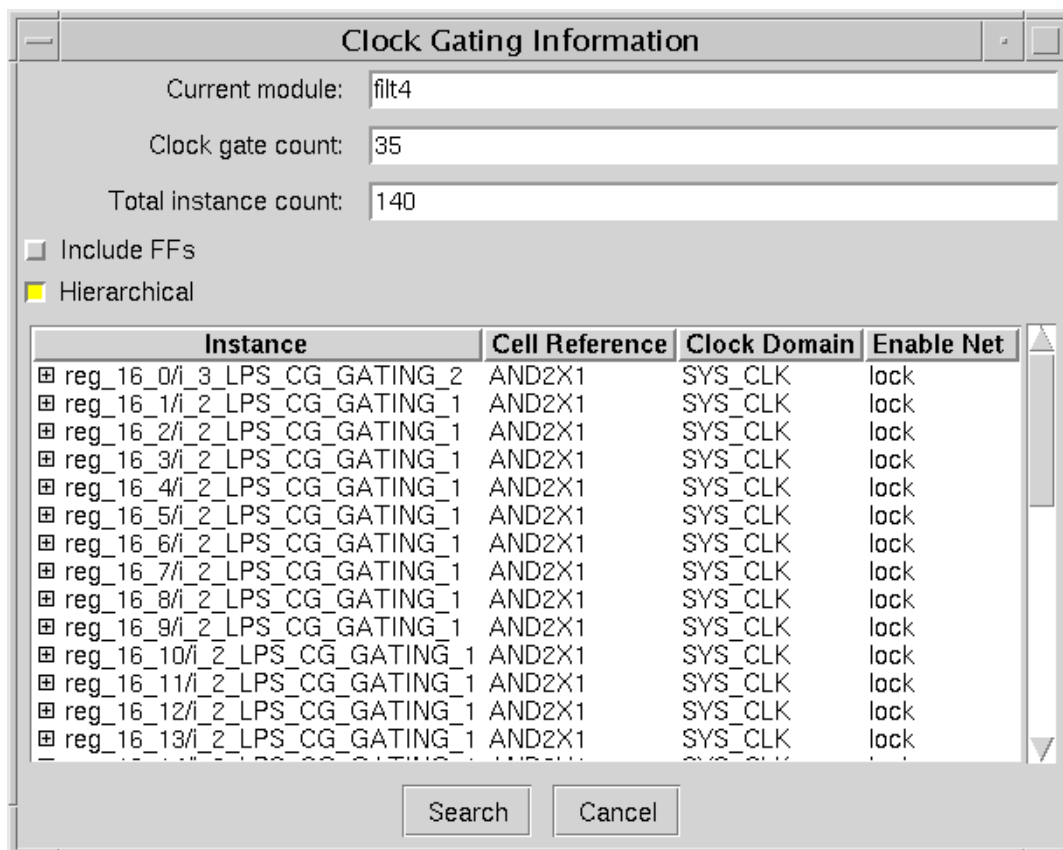
The Clock Gating Information window is displayed.

- c. Select *Hierarchical*.

- d. Click the *Search* button to populate the window.

The result is shown in Figure 5-8.

Figure 5-8 Clock-Gating Information Window



- e. Click left on any of the gating instance or component of the gating instance.

The instance is highlighted on the schematic.

- f. Click right on the same gating instance or component of the gating instance.

The hierarchical path is displayed in the clock-gating information window.

- g. Click *Cancel* to close the Clock Gating Information window.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

5. List the information on the clock-gating candidates in the console.

```
bgx_shell>set_current_module filt4
bgx_shell>get_gating_instance_list -hier
```

To make the output a little bit easier to read, you can enter:

```
bgx_shell>foreach i [get_gating_instance_list -hier] \
{puts [get_names $i]}
```

This returns a number of lines similar to the following one:

```
SYS_CLK LPS_CG_EN_1__0 lock i_2_LPS_CG_GATING_1
```

The first entry identifies the clock domain. The second one identifies the dummy module, the third one identifies the enable signal and the last entry identifies the gating instance.

Simulating and Reading the Toggle Count File

To do any meaningful power optimization, you need toggle count (or gate-level switching activity) information and to create this information in a Toggle Count Format (TCF), you must simulate the gate-level netlist.

1. Modify the testbench to create a TCF file.

Note: For this module, a modified testbench called `test_bench_0.v` is provided in the `simulation` directory, just for demonstration purpose. The test vectors are randomly generated.

You need to modify your testbench to use LPS Program Language Interface (PLI) tasks. The PLI tasks are provided in a shared library (which works fine with most Verilog simulators):

```
your_install_path/version/lib/archives/os/platform/lps_pli.so
```

- a. At the beginning of the time slot you want to start the capturing, insert:

```
$toggle_count(instance1, instance2 & );
```

- b. To stop recording the TCF file and specify the TCF file name, enter the following command to create a flat TCF file:

```
$toggle_count_report_flat(tcf_file_name, hier_top);
```

For a hierarchical TCF file, use the next routine which requires a hierarchical netlist:

```
$toggle_count_report_hier(tcf_file_name, hier_top);
```

After the simulation, a TCF file `tcf_file_name` is created with the toggle count information of the netlist.

The closer the testing vectors in the testbench represent the input vectors in a real situation, the more accurate the LPS results will be.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

Note: If you are a VHDL user, refer to “Generating TCF Files from a VHDL-Based Design” in *Low Power Synthesis for BuildGates Synthesis and Cadence PKS*.

2. Create the gate-level Verilog netlist needed for simulation.

```
bgx_shell>set_current_module filt4
bgx_shell>write_verilog -hier $sim_dir/filt4_lps.v
```

You see a warning message “Verilog continuous assignment written.” This is caused by assign statements dumped in the netlist. You can safely ignore that warning.

3. Start the simulation.

```
bgx_shell>verilog -f $sim_dir/sim_lps.opt
```

The `sim_lps.opt` file has the following content:

```
-v ../lib/tsmc18.v
+loadplil=lps_pli:lps_bootstrap
+toggle
+libext+.v
+notimingchecks
../simulation/test_bench_lps.v
../simulation/filt4_lps.v
```

The second line loads the LPS PLI shared library.

The `notimingchecks` switch turns off setup and hold time checks by the simulator. At this stage, the netlist is not fully timing-optimized yet and might contain some setup or hold time violations. If the Verilog simulator catches those violations, the simulation results might not be reliable, so you want to turn off the setup and hold time checks.

You see some warnings about too few port connections on gating latches, which you can ignore. The reason is that the latches have inverted outputs, which the dumped netlist did not characterize.

Note: While the simulation is running, you can read [More about TCF Files](#) on page 173.

4. Read the TCF file.

```
bgx_shell>read_tcf power_lps.tcf -scale 0.01
```

The command contains a scale factor. In the gate-level simulation you actually simulate with a clock 100 times slower to get a reliable result, which is a commonly used technique in gate-level circuit simulation. So when you read the TCF file in, you need to add a scale to adjust the toggle rate back.

Note: You can also read a TCF file from the GUI by selecting *File – Open*.

Optimizing the Power

Start the power optimization.

```
bgx_shell>do_optimize -power
```

In this final step of the basic LPS flow, BGX performs three tasks with one command:

- Deccommitting clock-gating logic

Note: LPS only decommits clock-gating instances if the proposed clock-gating logic does not save power or if it causes a timing violation.

- Committing sleep-mode logic

Note: LPS only commits sleep-mode logic if the sleep-mode logic saves power and does not cause a timing violation.

- Performing gate-level power transformations

During gate-level power optimization, LPS performs the following transformations to minimize the power dissipation in the design:

- Gate Sizing
- Pin Swapping
- Buffer Removal
- Gate Merging
- Slew Optimization
- Logic Restructuring

All these transformations are done automatically, so you cannot call any transformation individually.

Note: Depending on the performance of your machine, the optimization can run for a while. You can take this time to read [More about Gate-Level Power Optimization](#) on page 174.

Analyzing the Results and Generating Reports

1. Check the overall power dissipated in the current module.

```
bgx_shell>get_power
```

2. Get a further break down on the power dissipation.

```
bgx_shell>report_power
```

The result is shown in [Figure 5-9](#) on page 138.

Cadence Synthesis Rapid Adoption Kit Getting Started with Low Power Synthesis

Figure 5-9 Power Report

Report	report_power					
Options						
Date	20030527.161700					
Tool	bgx_shell					
Release	v5.10-s061					
Version	May 23 2003 12:12:42					
Module	filt4					
Timing Operating Condition	slow					
Power Operating Library	Typical					
Process	1.000000					
Voltage	1.620000					
Temperature	125.000000					
time unit	1.00 ns					
capacitance unit	1.00 pF					
power unit	1.00mW					

filt4						
		Library	Internal Cell	Leakage	Net	Total
Module	Instance	Cell	Power (mW)	Power (mW)	Power (mW)	Power (mW)
filt4			209.3068	7.607e-03	24.8811	234.1955

where

- ❑ Internal Cell Power is the total internal cell power for the current module (in mW).
- ❑ Leakage Power is the total leakage power for the current module (in mW).
- ❑ Net Power is the net power (in mW) of all nets in the current module.
- ❑ Total Power is the total power dissipated by the current module (in mW).

Note: To report the power usage on a none hierarchical instance, use the `report_cell_instance -power` command.

3. Scroll back the console to see what the tool has done to reduce power.

LPS first tried to meet the timing, then worked on clock gating committing or decommitting, then sleep mode commitment. Finally, LPS worked on gate-level optimization with the various techniques discussed in [More about Gate-Level Power Optimization](#) on page 174.

```
...
Info:    Committing clock gating of flip-flops < q_reg_0 q_reg_1 q_reg_10
q_reg_11 q_reg_12 q_reg_13 q_reg_14 q_reg_15 q_reg_2 q_reg_3
q_reg_4 q_reg_5 q_reg_6 q_reg_7 q_reg_8 q_reg_9 > in module
filt4/u4. The estimated power savings is 12.44%. <POPT-229>.
```

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

This message indicates that clock-gating logic in register bank `u4` got committed.

To report power on a selected object (net in this example), follow these steps:

1. In the Logical browser, double-click on instance `u4` to display its schematic in the Schematic window.
2. Click on the search icon (similar to a camera) in the Schematic window.
A Search Objects window pops up as shown in [Figure 5-10](#) on page 139.
3. Select *Net* under Object Type and type `lock` in the *Search Text*.
4. Click Search in the Search Objects window.

You can see the net `lock` in the Found Objects window.

Figure 5-10 Search for signal in Schematic Window



Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

5. Double-click on `lock`.

The net `lock` is now highlighted in red in the Schematic window.

6. Click right on the net `lock` in the Schematic window to bring up the Net pop-up menu.

7. Select *Power* in the pop up menu.

You get the following line in the console.

```
Net power (lock): prob(0.06250*), toggle rate(0.02500*), cap(0.25276),  
power(8.290000 uW) <NAVIGATES-106>.
```

The power report for the selected net lists the signal probability, the toggle rate (a ratio of toggle count over the total simulation time), the loading capacitance, and the total power dissipated on this net. The asterisk (*) indicates that the value comes from the TCF file, while a non * value would indicate that the value was computed.

You can get similar information by entering the following commands:

```
bgx_shell>set_current_module filt4  
bgx_shell>report_power -net u4/lock
```

Figure 5-11 shows the result. It reports the same information except that instead of reporting toggle rate, it uses the term transition density, which is the same as the toggle rate.

Figure 5-11 Report Power on a Net

filt4						
Module	Net	Probability	Trans. Den.	Capacitance	Power(mW)	
u4	lock	*0.0625	0.0250	0.2527	8.292e-03	

From the output of the `do_optimize -power` command, you can also see that one sleep mode logic got committed. If you `report_power -net` on signal `clipped` in instance `u6`, you will see the signal probability is about 0.29. As explained earlier in [Examining RTL Code of the round_clip Module](#) on page 128, the adder is shut off whenever the signal `clipped` is high, based on the following RTL code:

```
Assign data_out = clipped ? cap: rnd;
```

In this case, 29 percent of the time the signal `clipped` is high and LPS figured that the power consumption would be smaller if the sleep mode module got committed.

Power Analysis in BuildGates Extreme

You will use the following power analysis features offered by LPS.

- [Getting Statistical Reports](#) on page 141
- [Viewing The Power Level Legend](#) on page 143
- [Annotating Power](#) on page 144
- [Adding Color](#) on page 145
- [Displaying an Instance or Net Power Pie Chart](#) on page 146

Getting Statistical Reports

To get the statistical report on the probability (prod), enter the following command:

```
bgx_shell> report_tc_stats -prob
```

To get the statistical report on the transition density (td), enter the following command:

```
bgx_shell> report_tc_stats -td
```

Figure 5-12 shows the report for the transition density. The one for signal probability is similar.

Figure 5-12 Statistical Report on Transition Density

Transition Density (td) Statistics For Nets	
td	#nets
0.000000-0.000050	41
0.000050-0.000100	0
0.000100-0.000500	4
0.000500-0.001000	2
0.001000-0.005000	4
0.005000-0.010000	9
0.010000-0.050000	1349
0.050000-0.100000	1605
0.100000-0.500000	2762
0.500000->	625

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

You can also get a graphical statistical report that contains the switching activity on individual nets, as well as the overall distribution of the switching activity for a particular instance.

The toggle count histogram lets you view the probability distribution or toggle count information on a per net basis.

To display a toggle rate histogram, follow these steps:

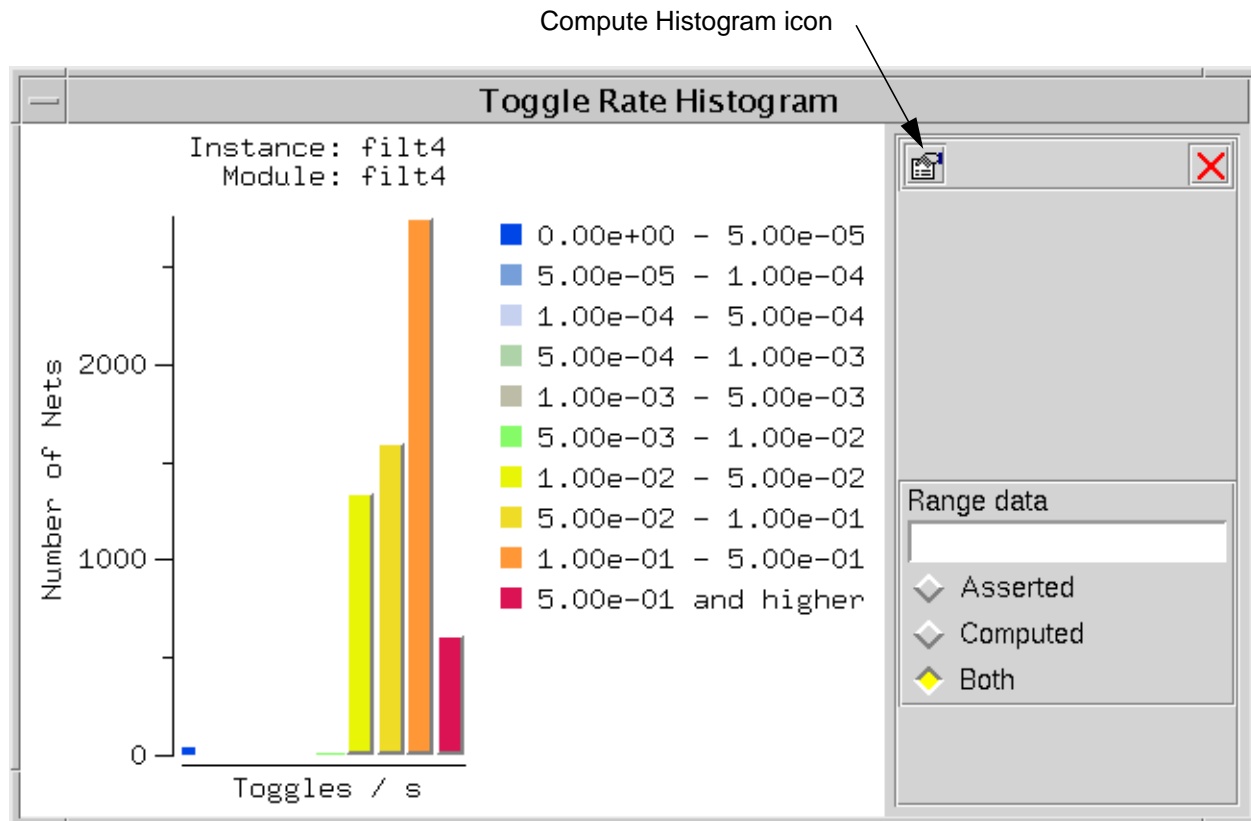
1. Make sure the current module is set to `filt4`.
2. Select *Reports – Power Toggle Rate Histogram*.

The Toggle Rate Histogram window appears.

3. In the Toggle Rate Histogram window, select the *Compute Histogram* icon as shown in Figure 5-13.

The Histogram shown in Figure 5-13 appears.

Figure 5-13 Toggle Rate Histogram



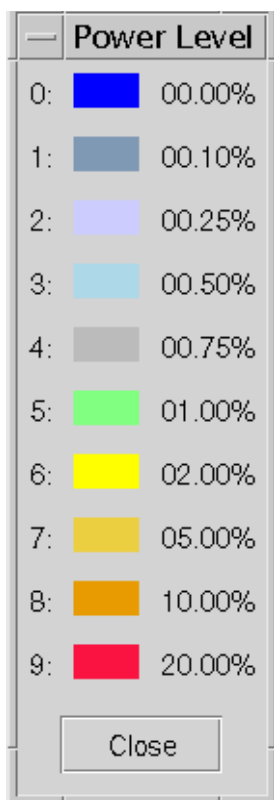
4. Close the Toggle Rate Histogram window by clicking on the Close Window icon.

Viewing The Power Level Legend

The Power Level Legend displays the colors used for the different power levels when annotating the power in the Schematic window or the Logical browser.

To display the Power Level Legend, select *View – Power Level Legend*. You see a window similar to the one shown in Figure 5-14.

Figure 5-14 Power Level Legend



The Power Level Legend has three columns:

- Level displays the ten power levels from 0 (the lowest) to 9 (the highest).
- Color displays the colors assigned to each level.
- Value displays the values assigned to each power level. By default, level 0 is 00.00% and level 9 is 20.00%.

To change the power level values and colors, select *Edit – Preferences – Colors*, and select the *Power Levels* tab.

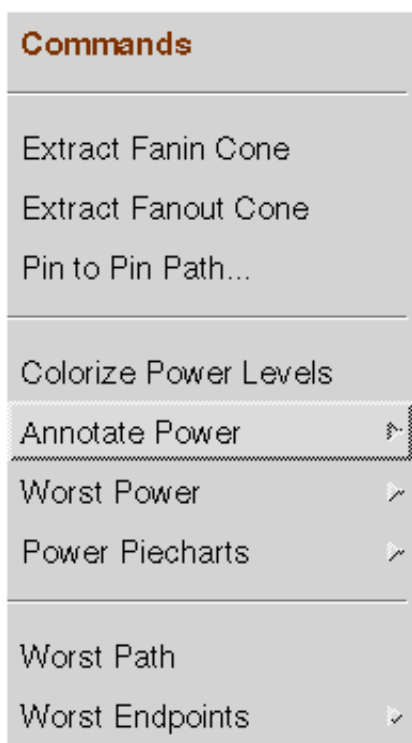
Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

Annotating Power

To annotate the power next to the instance or names in the schematic, follow these steps:

1. Select *Edit – Preferences – Schematic*.
The Schematic Preferences window appears.
2. Select the *Highlighting* tab.
3. Select *Power* under *Power Properties*.
4. Click OK.
5. Bring up the pop-up menu by clicking the right mouse button in the Schematic window.
6. Select *Annotate Power – Instance* or select *Worst Power*.



- *Annotate Instance Power* displays the power value below each instance in the schematic view. If you have a flat netlist, instance power is the sum of internal power and leakage power. For a hierarchical netlist, instance power is the sum of internal and leakage power of primitive cells in the hierarchy and power consumed by the nets driven by the cells.

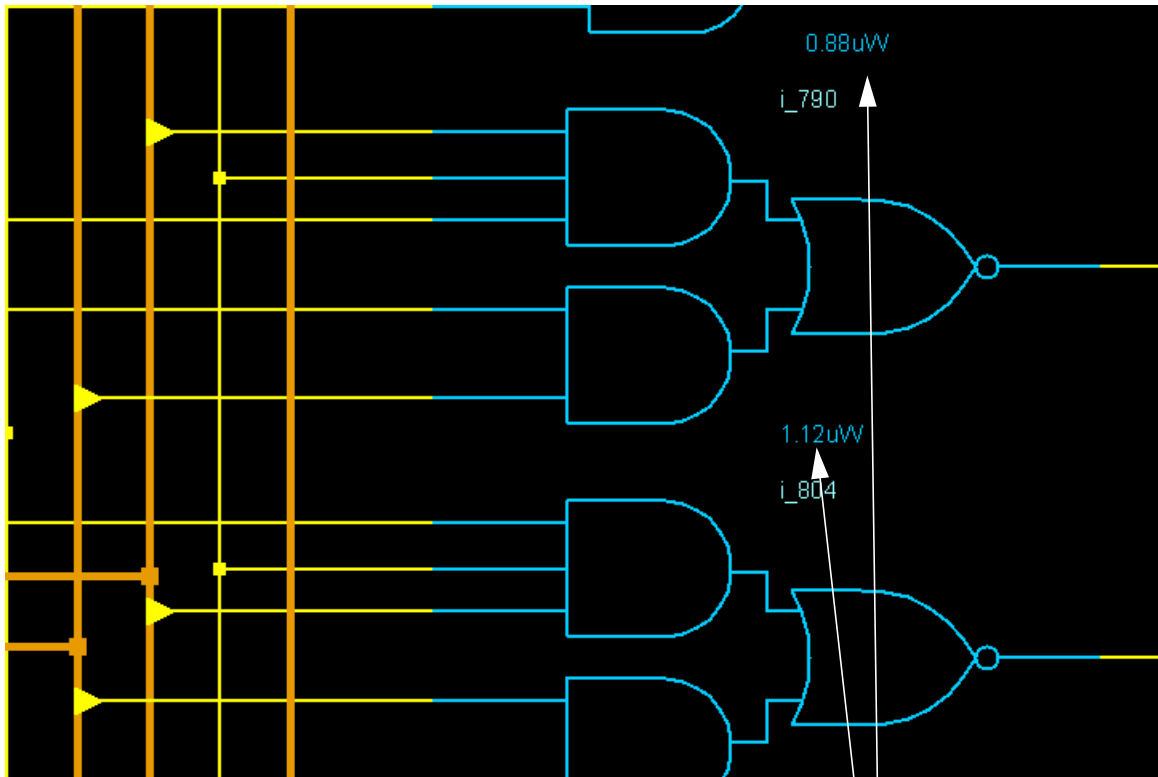
Figure 5-15 on page 145 shows the power consumption annotated for the instances. (You should zoom in several times to see this.)

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

- ❑ *Worst Power* highlights (in the Schematic window) the specified number of objects that contain the worst power values for the current module.

Figure 5-15 Annotated Power in Schematic



power numbers for instances

Adding Color

You can use color to display the power characteristics in the Logical browser and schematic.

To easier see the colorization, make sure to display the schematic on one page.

1. Select *Edit – Preferences – Schematic*.

The Schematic Preferences window appears

2. Select the Paging tab.
3. Set *Page Size* to *None*.
4. Click *OK*.

Cadence Synthesis Rapid Adoption Kit

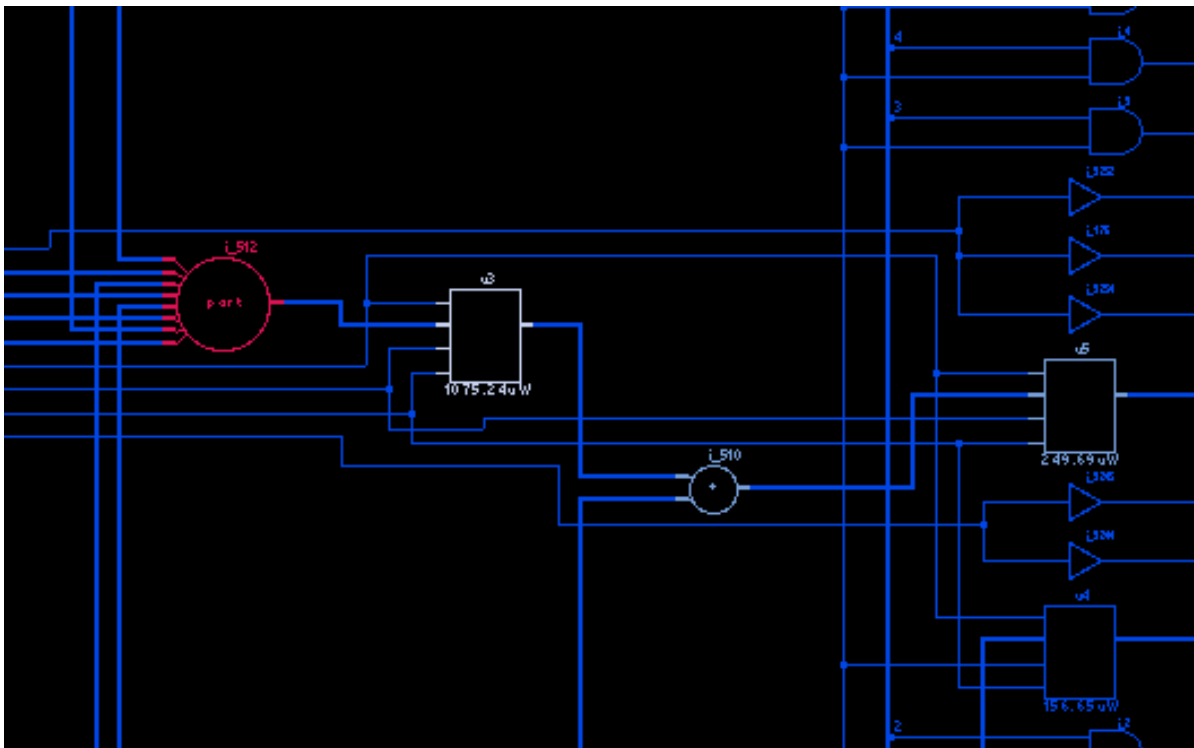
Getting Started with Low Power Synthesis

To add color to your schematic, follow these steps:

1. Bring up the pop-up menu by clicking the right mouse button in the Schematic window.
2. Select *Colorize Power Levels*.

The schematic is annotated with different colors. Figure 5-16 shows a zoomed-in part of the schematic. For the meaning of the colors, see [Viewing The Power Level Legend](#).

Figure 5-16 Colorize Power levels



Displaying an Instance or Net Power Pie Chart

To display an instance or net power pie chart, follow these steps:

1. Bring up the Commands pop-up menu by clicking the right mouse button in the Schematic window.
2. Select *Power Piecharts – Instance (or Net)*.

The Instance Power Usage window shown in [Figure 5-17](#) on page 147 appears. This window displays a pie chart of up to ten instances that consume the most power and the remainder. The Net Power Usage window shown in [Figure 5-18](#) on page 147 displays a pie chart of up to ten nets that dissipate the most power and the remainder.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

Figure 5-17 Instance Power Pie Chart

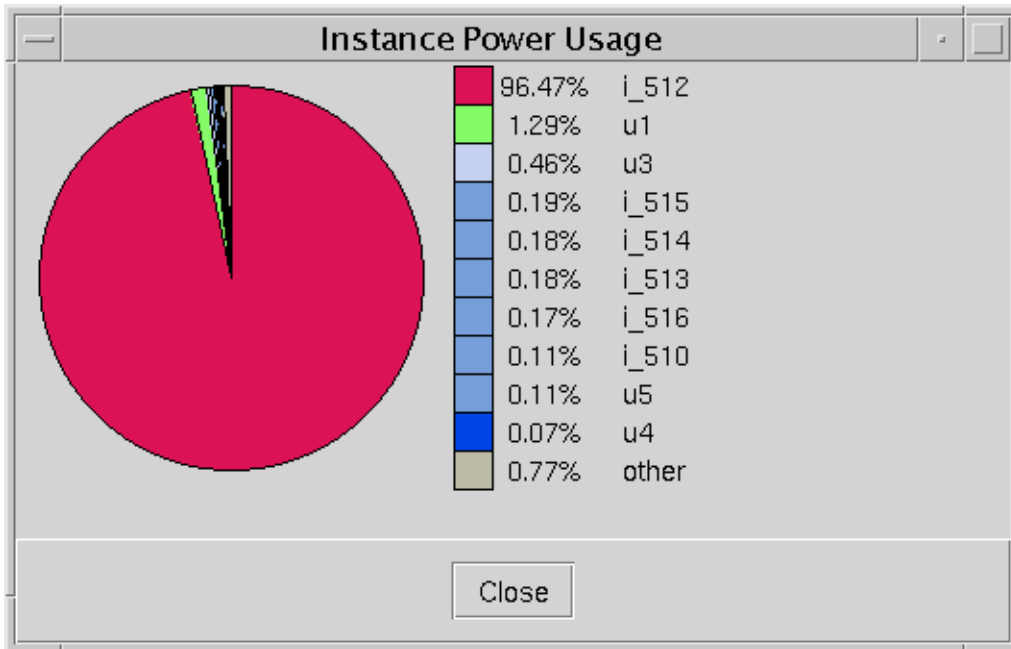
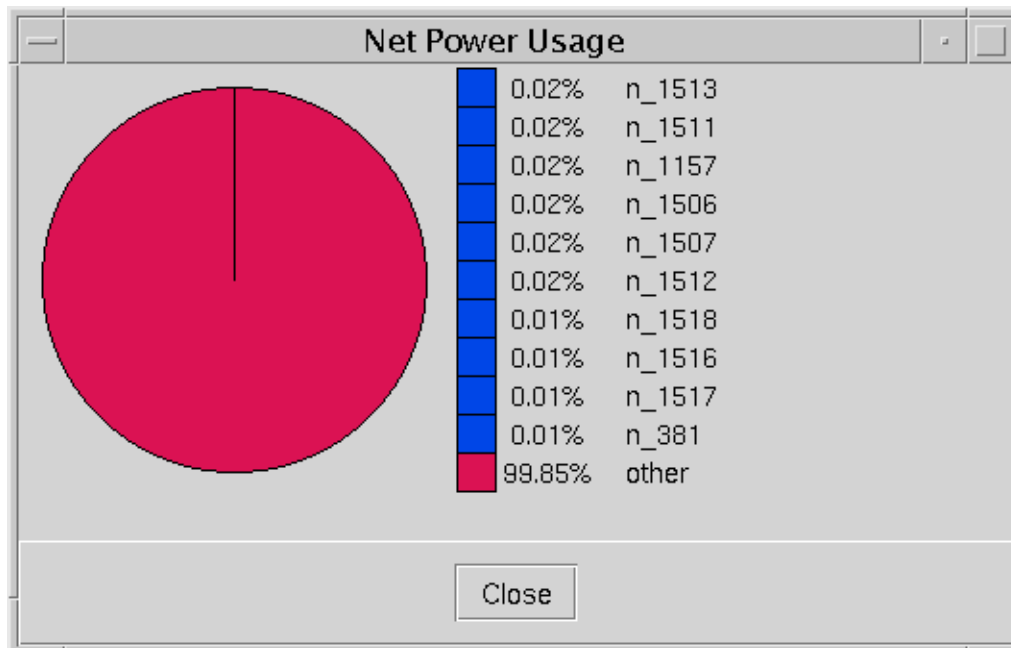


Figure 5-18 Net Power Pie Chart



3. Select *Close* on the Instance Power Usage and the Net Power Usage windows.

Generating Reports

1. Write out the design and the reports for timing, area, resources as well as power.

```
bgx_shell>write_adb -hier $adb_dir/filter_lps.adb
bgx_shell>write_verilog -hier $adb_dir/filter_lps.v
bgx_shell>report_timing > $rep_dir/filter_lps.rpt
bgx_shell>report_area >> $rep_dir/filter_lps.rpt
bgx_shell>report_resources -hier >> $rep_dir/filter_lps.rpt
bgx_shell>report_power >> $rep_dir/filter_lps.rpt
```

2. Compare the numbers you see in the `filter_lps.rpt` file to the ones in the `filter_no_lps.rpt` file.

Figure 5-19 Results Comparison with Normal Flow

	Area (um2)	Worst slack (ns)	Power (mW)
Normal flow	179449.31	+0.01	343.1023
LPS	165904.20	-0.00	234.1955

Note: Your results may differ from the ones shown here because of the randomly generated test vectors in the testbench used for the power simulation.

Writing Netlist and Database Files

1. Write out the Verilog netlist and database.

```
bg_shell> write_adb -hierarchical $adb_dir/filter_lps.adb
bg_shell> write_verilog -hierarchical $adb_dir/filter_lps.v
```

2. Select *File – Exit* or type `exit` in the console to end this session.

That concludes the regular LPS flow.



For maximum power accuracy, you should resimulate your design after the final `do_optimize -power` command.

The same applies to the sleep-mode only flow, clock-gating only flow and gate-level only flow, which are described in the next sections. For this tutorial, we did not rerun the simulation to save on run time.

Gate-Level Only Flow

Although LPS works best when starting from RTL, sometimes you might want to apply LPS to gate-level netlists only.

Sample Script Starting from a Gate-Level Netlist

```
read_alf                # or read_tlf
read_verilog            # or read_vhdl
do_build_generic
# Simulate
read_tcf
# Set timing constraints
do_optimize -power
```

Sample Script Starting from a Timing-Optimized Netlist

```
set_global depth_for_fast_slew_prop 2
read_alf                # or read_tlf
read_verilog            # or read_vhdl
do_build_generic
# Simulate
read_tcf
# Set timing constraints
do_xform_optimize_power -gate_level
```

Note: The value for the global `depth_for_fast_slew_prop` is set to 2 to prevent slack from getting worse.

The `run_gate.tcl` script (which is provided in the `tcl` directory) starts with a timing-optimized netlist for this filter design and allows you to run this flow.

LPS-DFT flow

The following is a sample DFT-LPS observability flow script:

```
read_alf                # or read_tlf
read_verilog            # or read_vhdl
set_global dft_scan_path_connect tieback
do_build_generic -sleep_mode
# set timing constraints
check_dft_rules
set_sleep_mode_options
set_clock_gating_options
do_optimize -power -stop_before mapping
check_dft_rules
do_optimize -power -stop_for_power_simulation
# simulate
read_tcf
do_optimize -power
set_global dft_scan_path_connect chain
do_xform_connect_scan
```

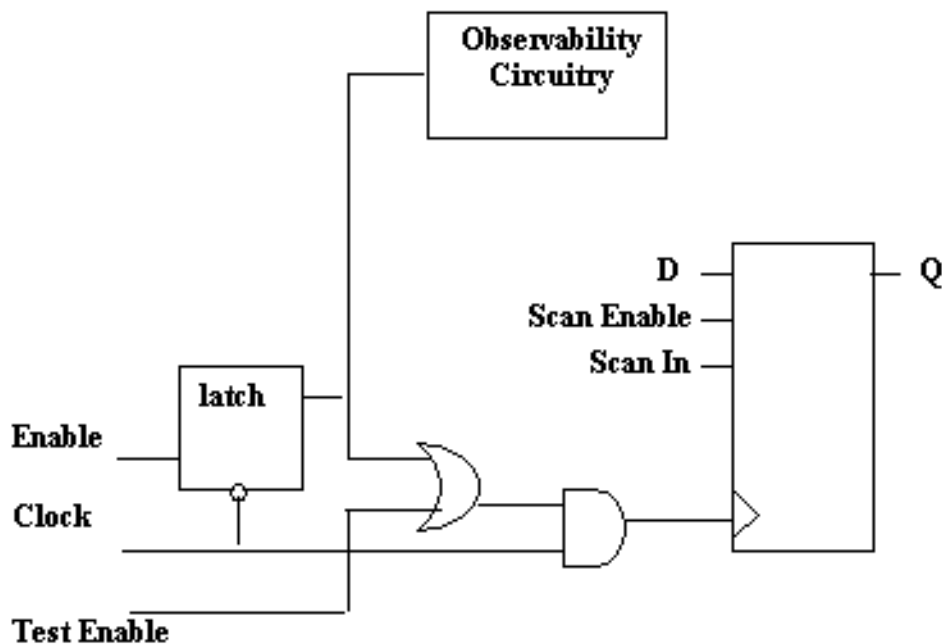
Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

The `run_lps_dft.tcl` script (which is included in the `tcl` directory) allows you to run this flow on the `filt4` design. LPS can add controllability and observability to the design and is fully compatible with DFT. BuildGates actually allows you to make a design with multiple clock domains observable. For more information about using DFT, see [Design for Test \(DFT\) Using BuildGates Synthesis and Cadence PKS](#).

Figure 5-20 illustrates how LPS adds a control port and observability circuit to a register.

Figure 5-20 LPS Observability Circuitry



You can add a control port and observability circuit to a register, using the following single command:

```
bgx_shell>set_clock_gating_options -control post_seq_element \  
-auto_test_port true \  
-observe true \  
-domain clock_net \  
-obs_style port
```

`control post_seq_element` adds the controllability logic after the sequential element. In the circuit in Figure 5-20, the OR gate is inserted after the latch.

`-auto_test_port` creates a test port in the top module. If you do not want an additional port in your design, you can either include the observability register in a scan chain or share the test ports with functional input/output port.

Cadence Synthesis Rapid Adoption Kit

Getting Started with Low Power Synthesis

`-observe` makes all gated signals observable. Each of the gating signals is added to an XOR tree and the output of the XOR tree is connected to either a port or observability register. In this case, a port will be used since you specified `-obs_style port`.

`-domain` groups gating signals driven by the same clock signal under DFT settings.

`-obs_style` creates a port in the top module for each observability domain and its logic is connected to this port through an XOR tree.

Summary

In this session, you

- Became familiar with the LPS design flow which provides power estimation, and optimization techniques that consider timing constraints.
- Learned that LPS makes all power decisions based on actual power computations rather than rule based decisions.
- Learned that LPS minimizes the power by committing the RTL transformations and performing gate-level power optimizations while satisfying the timing constraints
- Learned various techniques to analyze the power saving results.
- Received an introduction to the DFT-LPS flow

Cadence Synthesis Rapid Adoption Kit
Getting Started with Low Power Synthesis

More about BuildGates Synthesis

- [More about Optimizing the Design](#) on page 154
- [More about Timing Reports](#) on page 156

More about Optimizing the Design

Top-Down Optimization

In top-down design flow, the designer sets the constraints at the top level, and optimization is performed from the top level without first individually optimizing lower-level modules. Top-down flow is traditionally used to optimize smaller designs. However, BuildGates Synthesis has much higher capacity. Compared to other tools, it can handle much larger designs top down. You can easily do top-down optimization on one million gate designs. BuildGates Synthesis top-down flow is very easy to use, and it doesn't require complicated scripting and typically produces a better quality of results (QOR).

Bottom-Up Optimization

In the bottom-up optimization approach, lower-level modules are first optimized individually with constraints derived from top-level constraints. These optimized lower-level modules are then “stitched together” and marked `dont_modify`. Then, the next level of the hierarchy is optimized until all levels including the top level are optimized. The process of deriving lower-level constraints from top-level constraints is commonly referred to as time-budgeting. The `do_time_budget` command in BuildGates appropriately splits combinational delay requirements across module boundaries based on the number of levels of logic and compressibility of the logic. Compared to other tools, the BuildGates Synthesis time budgeting process involves fewer iterations and requires less scripting time from the user.

More on `do_optimize`

The `do_optimize` command is the simplest way to optimize the design. It applies to the hierarchy in and under the current module, as set by the `set_current_module` command. Depending upon the state of the design database, logic optimization will include some or all of these steps: uniquification, constant propagation, structuring, redundancy removal, technology mapping, timing-driven optimization, buffering of multiport nets, and design-rule fixing. All of these steps can be run individually or as a subset using a unique concept called transforms. These transforms have a prefix of `do_xform_`. Transforms are a very powerful means of customizing your optimization flow. [Figure A-1](#) on page 154 shows the `do_optimize` flow.

Figure A-1 `do_optimize` Flow

Cadence Synthesis Rapid Adoption Kit

More about BuildGates Synthesis

Return to [do_optimize Info Messages](#) on page 43 for more information on the messages generated by `do_optimize`.

More about Timing Reports

Analyzing Simultaneous Best Case and Worst Case Analysis

Setup checks are performed using worst case PVT corner(s) while hold checks are performed using best case operating corner(s). In the simplest of the cases, you specify two PVT corners (best and worst) and setup, hold checks are done as mentioned.

There are several ways to set up BuildGates for simultaneous best case, worst case analyses.

Example for Best Case-Worst Case Analysis Using One Library and Two Operating Conditions

```
set_operating_condition -pvt min bccombest
set_operating_condition -pvt max wccomworst

set_global pvt_early_path min
set_global pvt_late_path max
set_global timing_analysis_type bc_wc
```

In this case, setup checks are performed using the worst case operating condition and the hold checks are performed using the best case operating condition.

Example for Best Case-Worst Case Analysis Using Two Libraries and Two Operating Conditions

```
read_alf -min min_lib -max max_lib -name merged_library_name

set_operating_condition -library min_lib -pvt min bccom
set_operating_condition -library max_lib -pvt max wccom

set_global pvt_early_path min
set_global pvt_late_path max
set_global timing_analysis_type bc_wc
```

Setup checks are performed using the worst-case operating condition. The worst-case delays are picked from `max_lib` and scaled appropriately for the worst operating condition. Similarly, the hold checks are performed using the best-case operating condition. The best-case delays are picked from the `min_lib` library and scaled appropriately for the best operating condition.

Return to [Generating Area Reports](#) on page 50.

More about Datapath

- [More about Operator Merging](#) on page 158
- [More about Automatic Architecture Selection](#) on page 161
- [More about the AmbitWare Library](#) on page 163
- [More about Extended Languages](#) on page 166

More about Operator Merging

The goal of the BGX datapath feature is to provide powerful optimization of datapath streams and components, but in a standard, easy to use synthesis flow, and with automatic timing closure between datapath and control logic. After automatically partitioning the datapath and control logic, several powerful datapath techniques are applied to produce a handcrafted-quality result. These include operator merging, automatic architecture selection, and architecture refinement techniques. For details on how each of these features work, refer to *Datapath for BuildGates Synthesis and Cadence PKS*.

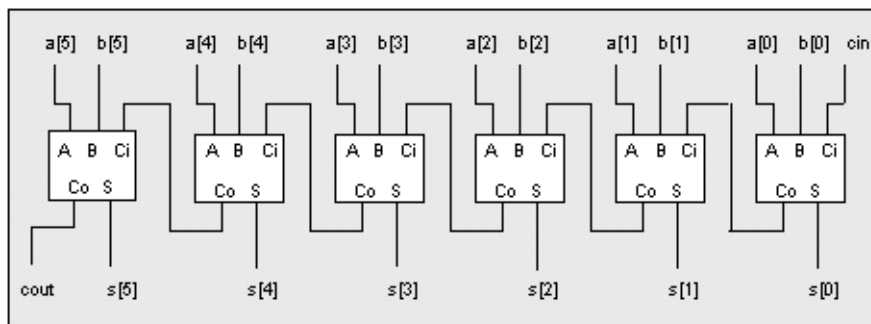
Operator merging is a powerful technique that eliminates redundancies that naturally exist between sequentially connected discrete datapath components. It will improve both speed and area QOR. In general terms, what operator merging does is first locate a datapath component (one of 13 functions including arithmetic and comparisons).

Next, it reads upstream and downstream from that component and locates other sequentially connected components. Sequentially connected datapath components have naturally occurring redundancies and lend themselves well to optimization.

Finally, operator merging groups these components, determines their function, and replaces them with a custom build component with the same function. This custom built component has several key features that make it smaller and faster than the previous discrete components.

For instance, in this example, each discrete component in the original design has a carry propagate adder (CPA): See Figure B-1.

Figure B-1 Carry Propagate Adder (Ripple Adder)



The most basic CPA is a ripple adder (shown). The ripple adder has the disadvantage that the critical path is *dependent* on the bit width because it follows the ripple action from the least significant bit (LSB) to the most significant bit (MSB). Thus, a 128-bit graphics controller, whose library has full adders with 0.5 ns of delay would be limited to a speed of 64 ns or

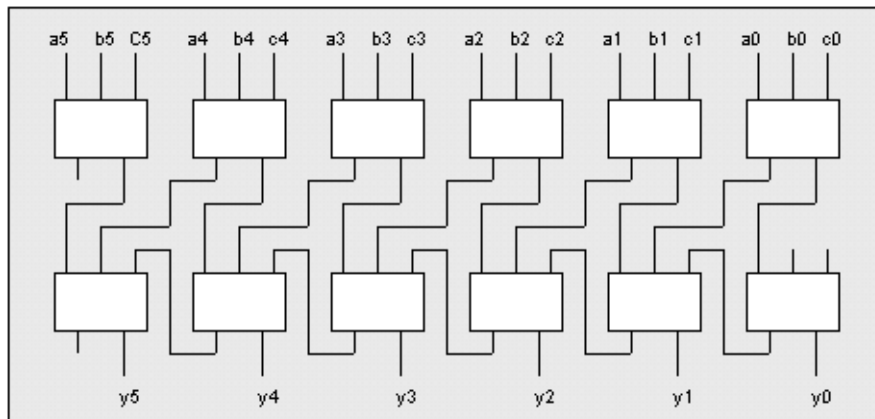
Cadence Synthesis Rapid Adoption Kit

More about Datapath

15.6 MHz if a ripple adder was used. The disadvantage of having CPAs is that, if they are in a timing critical path it is very expensive (in terms of area) to speed them up. Off the critical path, it is no problem to go with a ripple adder because it is small.

Operator merging solves this problem by converting each of the intermediate CPA adders to a format known as 3:2 compression illustrated by Figure B-2.

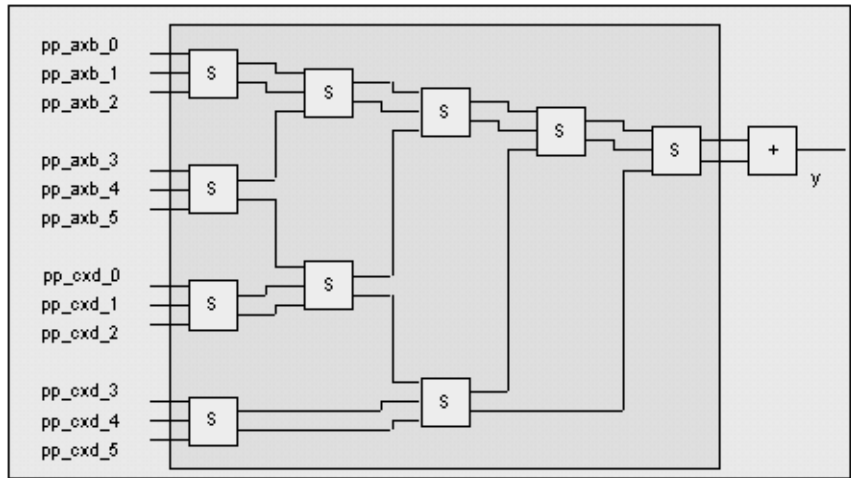
Figure B-2 Compression



The top adder is in 3:2 compression and the one below is a traditional CPA adder. Notice that the primary inputs of the first adder have been used, including the carry bit for addition. Also notice that the carry bit was passed to a downstream adder. 3:2 compression has the same number of components and does the same amount of computation—it just does it in a different order. By changing the order of the computation, the 3:2 compression improves the speed of the path.

After converting over each of the intermediate adders to 3:2 format, operator merging adds these 3:2 adders to a Wallace Tree. The second technique that operator merging applies is a Wallace Tree optimization. Through this optimization, the tool orders the inputs to the Wallace Tree according to the arrival times. This also includes streamlining the critical path to the output of the Wallace Tree. Effectively, this means that the critical path can almost *bypass* the adders that were previously in its path. Once again, this substantially improves the speed of the path. [Figure B-3](#) on page 160 shows an example diagram of a Wallace Tree:

Figure B-3 Wallace Tree



The last step in the operator merging technique is to select a final adder for this custom-built module that is just large enough to meet the speed requirement of the design. Quite often, so much improvement is made with 3:2 compression and Wallace Tree optimization that smaller adders can be used. This feature, in conjunction with the elimination of slow CPA adders from the critical path, results in area savings for the design as well as increased speed.

Note: Operator merging works best if you do not include a lot of hierarchy between datapath components, because the tool defaults to preserving a user-defined hierarchy. Therefore, do not place each multiplier in your design in a module by itself. Operator merging only applies to inferred components, not to gate level netlists or to instantiated components.

Return to [Operator Merging](#) on page 108.

More about Automatic Architecture Selection

Automatic Architecture Selection (AAS) is a technique that automatically selects the correct architectures for your adders and multipliers in your design.

First it looks at the timing context of the module. For example, off the critical path, area-optimized adders are selected. On the critical path, adder size and speed is experimentally increased until timing is met or the largest adder is used.

Next, AAS looks at the bit-width context and library availability when selecting architectures. For example, for multipliers, BGX considers the bit-width of the operands to determine if it makes sense to build a booth-encoded or non-booth multiplier. Also, it tries to use specialized cells in the library, such as booth cells or 4:2 compression cells. If those are not available, the tool builds the cells it needs. In another example, the tool uses a square instead of a multiplier if it finds that the multiplier and multiplicand are the same vector.

The AAS technique is applied to every datapath operator in your design, including the inferred and instantiated operators, as well as the customized operator-merged components. For more on AAS, refer to the *Datapath for BuildGates Synthesis and Cadence PKS*.

The tool also does automatic timing-driven implementation refinement. For example, if all inputs have the same arrive time, the tool may implement what is shown in Figure B-4. This reduces the levels of logic from input to output and should lead to the best timing. However, if the input arrival time is skewed, as show in Figure B-5, the software adjusts the order of addition accordingly, from iteration to iteration, to achieve the best timing.

Figure B-4 Timing-Driven Implementation Refinement with Uniform Arrival Time

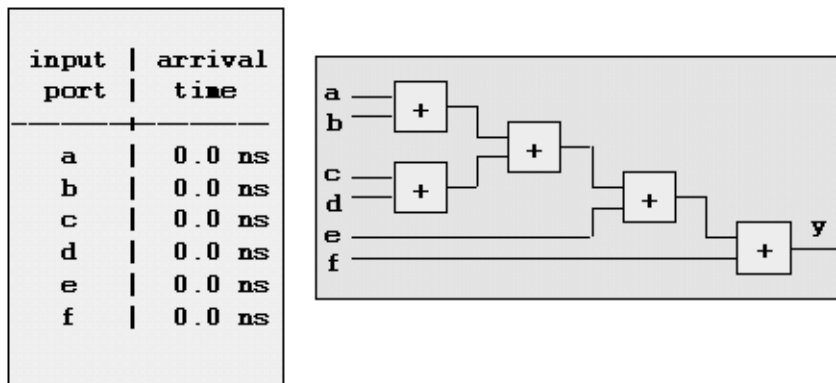
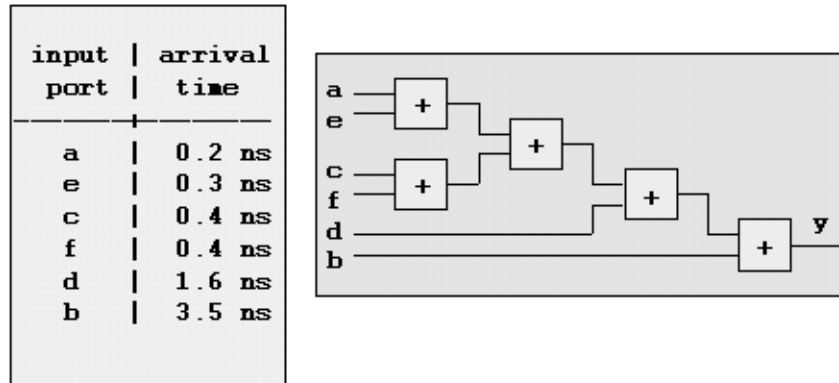


Figure B-5 Timing-driven Implementation Refinement with Skewed Arrival Time



If this adder tree is part of a bigger design and the input skew must be derived from the surrounding logic, it is hard to manually predict the skew and decide the configuration and order of the adder tree. An adder tree like this can be found as part of the carrysave reduction tree inside of a multiplier, where timing from the surrounding logic is very difficult to calculate manually. The software is better equipped for the job because it can calculate the timing information dynamically.

Return to [Generating Reports](#) on page 111.

More about the AmbitWare Library

Another feature of the BGX datapath offering is the free AmbitWare component library. This library contains many of the popular datapath components that are used in complex networking, multi-media, DSP, and processor designs today.

BuildGates Synthesis comes with three pre-defined AmbitWare libraries that provide both synthesis and simulation models of all of the components:

- **AmbitWare Complex Arithmetic Library (AWARITH)**—Defines complex arithmetic functions in RTL that can be instantiated in the source RTL. For example, a combined incrementer and decrementer, multiply accumulators, pipelined multipliers, and vector adders.
- **AmbitWare Complex Logical Library (AWLOGIC)**—Defines complex logic functions that can be instantiated in the source RTL, for example, arithmetic and logical shifters, rotaters, encoders and decoders.
- **AmbitWare Sequential Logic Library (AWSEQ)**—Defines complex sequential logic functions that can be instantiated in the source RTL. For example, a reset-enable flip-flop, and a n-tap shift register.

The three AmbitWare libraries use the IP model by containing the encrypted models of components. The synthesis models of the components are defined in Verilog or VHDL RTL, the simulation models of the components are defined in Verilog and VHDL. The components are then decrypted, synthesized and optimized in the context of the design.

Below are some examples:

- Pipelined Multiplier
- Multiply-Accumulate (MAC)
- Vector Add
- Comparator
- Square
- Combinational Divider with Remainder
- Barrel Shifter
- Programmable Incrementer/Decrementer
- Programmable Adder/Subtractor
- Absolute Value

- Priority Encoder/Decoder
- Rotate Left/Right
- Generalized Sum of Products
- Saturator
- Rounder
- Overflow Detector
- Blender
- Universal Multiplexer
- Register with Reset and Enable

The way to access these components is through instantiation, which involves specifying the component parameters and mapping the ports. Figure [B-6](#) shows an example.

Figure B-6 Instantiation of a Pipeline Multiplier

```
module fun (A, B, TC, CLK, Y);
input [8:0] A;
input [6:0] B;
input TC; // 0 if unsigned, 1 if signed
input CLK;
output [15:0] Y;
// 9x7, 1 pipeline stage added
AWARITH_PIPEMULT # (9,7,1) U0
(.A(A), .B(B), .TC(TC), .CLK(CLK), .Z(Y));

endmodule
```

The port map is shown for a pipeline multiplier. Each of the components comes with a simulation model as well several examples of how to use these components (see the [*AmbitWare Component Reference*](#)).

Each component used from the library is custom built for the particular usage. The construction is similar to the optimizations done on operator-merged components. For example, the final adders in the pipeline multiplier above are selected based on the timing context of the design. It is important to note, however, that instantiated components are not eligible for operator merging.

Inference Verses Instantiation

For several datapath components, you have the option to infer them or instantiate them from the library (see Figure B-7). This is similar to Synopsys. You could use $y = a + b$; or you could instantiate an adder `DW01_ADD (.A (A)...)` and so on.

Figure B-7 Difference between Instantiation and Inference

```
// The following is an inference to an adder:  
y = a + 1;  
  
// This is an instantiation of an incrementer:  
AWARITH_INCDEC #(width,0) U0(.A(a), .DEC(1'b0), .Z(y));
```

For the most part, with BGX, the result will be the same for inference as with instantiate, with the exception that instantiated components are not eligible for operator merging. Because operator merging offers improved QOR, you should probably use inference whenever possible. See the *[Datapath for BuildGates Synthesis and Cadence PKS](#)* for details.

Return to [Introduction to BGX Datapath](#) on page 100.

More about Extended Languages

The following is the relationship between various Verilog languages:

- Verilog-2001 (originally named Verilog-2000) is a superset of Verilog-1995.
- Verilog-DP is a superset of Verilog-2001.
- Verilog-1995 and Verilog-2001 are both IEEE standards.

BGX supports all three versions. Verilog 2001 and Verilog-DP are *purely additive* extensions of Verilog-95. This means that you do not have to convert your design over to take advantage of the new features. Verilog-2001 contains several enhancements to the Verilog-95, which makes designing with Verilog easier. Of particular interest are the generate function and the signed data type.

Previously, in Verilog-95, sign extension was a manual effort. It was fairly laborious and somewhat cryptic. The truncation and part-select involved in this created some limitations for advanced datapath engines. By supporting the signed data type, these limitations are lifted and the code becomes much cleaner. Moreover, if your design uses only the unsigned data type, BGX can accomplish the same quality of results as using signed data type, because it understands the signed nature behind the RTL coding.

Verilog-DP contains several datapath specific enhancements to Verilog-2001. Unlike MCL, a proprietary language from Synopsys, Cadence has asked the IEEE to approve Verilog-DP as a standard. Also there is another salient difference: Verilog-DP is *purely additive*—you do not have to re-code your design to take advantage of the features it offers. Simply use your design as it exists today and change only the parts where you want to take advantage of the new features.

Here is a list of some of the compelling features of Verilog-DP:

- New signal attributes and querying functions:
 - Signed (Verilog 2001)
 - carriesave (via pragma)
 - Signal attribute querying functions (`$size`, `$is_signed`, `$low`, `$high`)
- Attribute inheritance
 - Signal attributes can be deferred and inherited from the parent module via signal connection
 - Allows highly parameterized designs without explicit parameters

Cadence Synthesis Rapid Adoption Kit

More about Datapath

- Explicit replication and conditional compilation
 - `for`, `if` and `case` (compatible with Verilog 2001)
- Array Signals and Operations
 - Array signals and I/O ports
 - Array operations: vector inner-product, vector-matrix product and other linear algebra operations
 - Array initialization, flattening and unflattening functions
- New Datapath Primitives
 - Succinct representation and improved QOR
 - `$blend` (linear interpolation), `$abs`, `$sgnmult`, `$lead0`, `$lead1`, `$sat`, `$rotatel`, `$rotater`, `$iroundmult`, `$itruncmult`

Return to [Required Files](#) on page 102.

Cadence Synthesis Rapid Adoption Kit
More about Datapath

More about Low Power Synthesis

- [More about LPS Clock Gating Technique](#) on page 170
- [More about TCF Files](#) on page 173
- [More about Gate-Level Power Optimization](#) on page 174

More about LPS Clock Gating Technique

Depending on the performance of your machine, the synthesis can run for a while. You can take this time to read more about the clock-gating techniques used by LPS.

In LPS, clock-gating consists of the following two steps:

1. Exploration and insertion:

LPS starts with identifying the set of registers for clock gating and the periods of inactivity in those registers.

Next, LPS inserts gating logic for clocks in the inactive periods identified during the exploration phase.

2. Commitment

During this phase, LPS commits gating logic based on power savings, performs timing checks for the setup and hold times of the gated clocks, and removes gating logic where timing constraints are violated or if there is no possible power savings.

Even though data is loaded into registers very infrequently in many 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.

By using clock gating, power is not consumed during the idle period when the register is shut off by the gating function. Also, the logic of the enable circuitry on the data pin in the original design is removed which saves power.

For example, consider the following Verilog module excerpted from the `filt4.v` file.

```
module reg_16 (clk, lock, set, d, q);
    input clk, lock, set;
    input [15:0] d;
    output [15:0] q;
    reg [15:0] q;
    always @ (posedge clk or posedge set)
    begin
        if (set == 1'b1)
            begin
                q <= 16'b0;
            end
        else
            begin
                if (lock == 1'b0)
                    begin
                        q <= d;
                    end
            end
    end
end
```

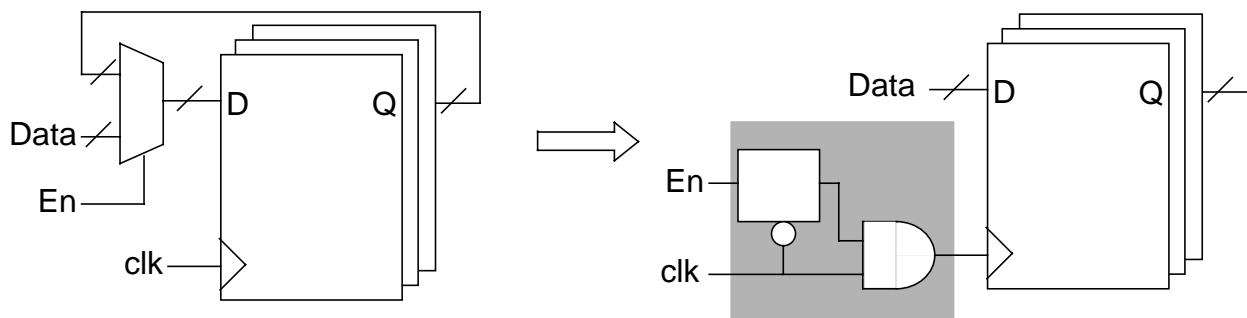
Cadence Synthesis Rapid Adoption Kit

More about Low Power Synthesis

```
end
end
endmodule
```

reg_16 is register bank with an enable signal ~lock. Figure C-1 illustrates how clock gating works to reduce power.

Figure C-1 Clock-Gating Example

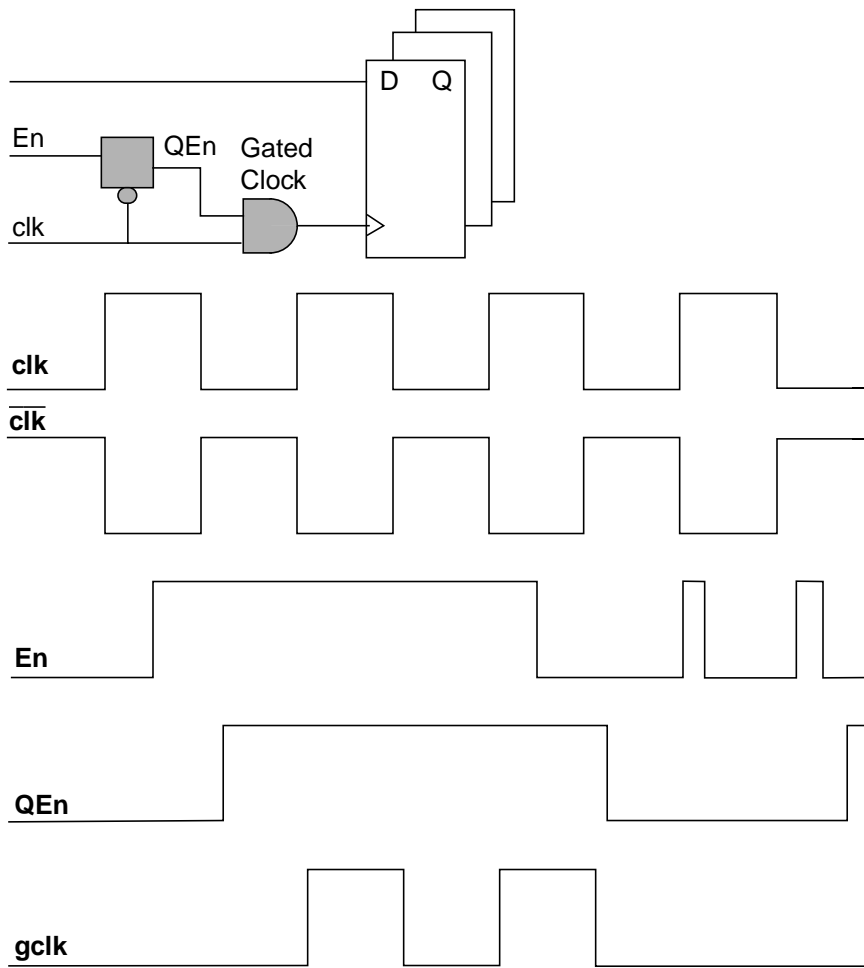


Instead of adding a MUX in front of each data input of every register, LPS gates the clock with an AND gate to drive the entire register bank. You can control the number of fanouts that the clock-gating cell output can drive using the `set_clock_gating_options -max_fanout` command. If `-max_fanout` is set to 3 and 6 fanouts are required, LPS clones the gating cell.

Figure C-1 on page 171 also shows that a latch is inserted before the AND. This latch is used to eliminate any possible clipping on the gated clock signal. Without the latch, a spike on signal `lock` can propagate through the AND gate and induce clock clipping, which is bad. If you are sure the enabling signal is glitch free you can eliminate the latch using `set_clock_gating_options -gating_style none`.

Figure C-2 on page 172 illustrates how latches can eliminate spikes on an enabling signal.

Figure C-2 Eliminating Spikes with Latch



Here is how timing check is done on gated logic. The data input of the gating cell must be stable:

- Before the clock input makes a transition from the controlling value to the non-controlling value (setup time).
- After the clock input returns to a controlling value (hold time)

You can change the setup and hold times using the `set_clock_gating_check` command.

With a Physically Knowledgeable Synthesis (PKS) license, LPS can save even more power with the root gating technique. In this case, LPS inserts gating logic at the root of the clock to save the power consumed by the clock-tree network. This is in addition to gating the register banks. For more details, see [Root Gating in Low-power for BuildGates® Synthesis and Cadence® PKS](#).

Return to [Optimizing the Design](#) on page 132, step 2.

More about TCF Files

A Toggle Count Format (TCF) file contains net or pin switching activities, as well as signal probabilities. TCF is not an industry standard format, but a pure Cadence format. The TCF file can be flat or hierarchical. The switching activity equals to the toggle counts on the net or pin during the entire simulation. The signal probability means the probability for this signal to stay at 1 in the total recorded toggle counts.

If none of the logic nodes in the fanin cone contain assertion values, LPS assumes default values on the primary inputs and outputs of the sequential elements. The default value of signal probability is 0.5, and the default value of the toggle count is half of the clock toggle count. If there is no clock in the design, the transition will be simply 0. For internal nets, LPS uses probabilistic technique to the switch activities. New nets might be inserted in optimization, and if there is a request for the switch activities on any of those newly created nets, LPS uses probabilistic technique to calculate the switch activity for those nets upon request.

Return to [Simulating and Reading the Toggle Count File](#) on page 135, step 4.

More about Gate-Level Power Optimization

- [Gate Sizing](#) on page 174
- [Pin Swapping](#) on page 176
- [Buffer Removal](#) on page 176
- [Gate Merging](#) on page 177
- [Slew Optimization](#) on page 178
- [Logic Restructuring](#) on page 179

These LPS optimization techniques work with the timing engine to identify trade-off points in the power delay curve. When combined with the various effort levels, you can choose between faster run time or a better quality result in terms of power dissipation.

Gate Sizing

Gate sizing makes the CMOS gate size smaller or larger to minimize the total power dissipation without violating timing constraints.

In general, reducing the size of a gate leads to a decrease in power dissipation but to an increase in delay.

The overall internal cell power dissipation of a gate consists of:

- Leakage power
- Short-circuit power
- Power dissipated by parasitic capacitance

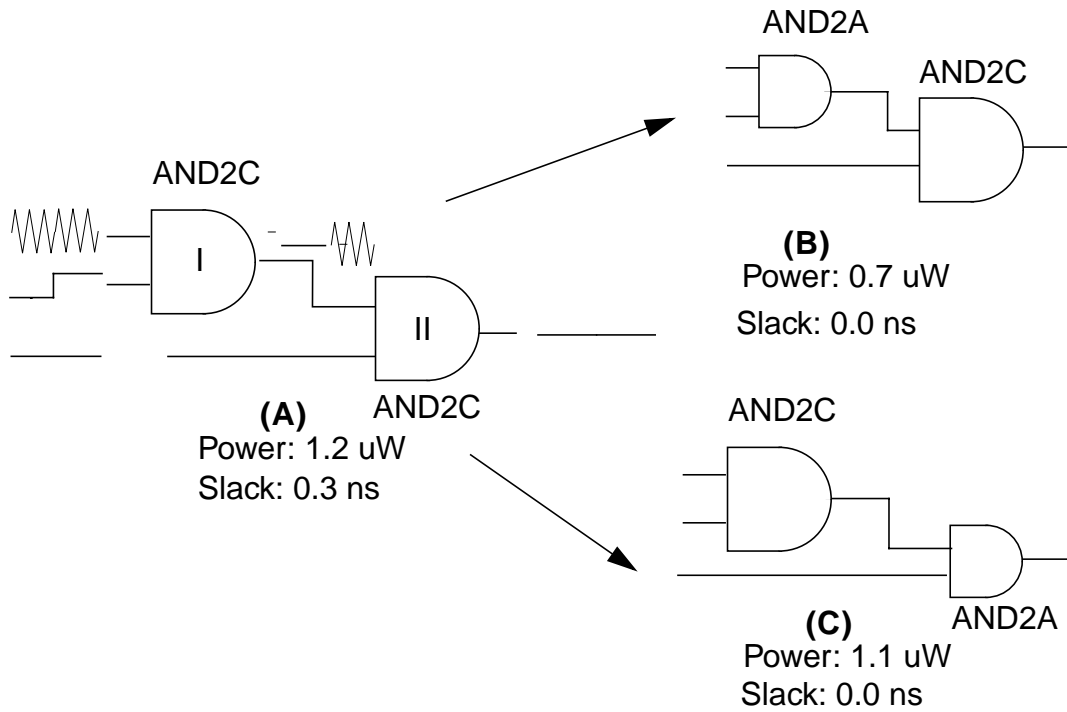
Sizing the gate reduces the short-circuit power dissipation and power dissipated by parasitic capacitance. Gate sizing can, however, increase the short-circuit power dissipation of the fanout logic because the slew rate of the output signal can increase when this transformation is performed. LPS makes tradeoffs between the gate sizing and the increase of power dissipated due to slew rate in order to optimally reduce the power dissipation of the design.

Figure [C-3](#) demonstrates the importance of gate selection for gate sizing to maximize power dissipation savings. The power of the circuit in Figure [C-3](#) (A) can be reduced by downsizing either gate I or gate II, but not by both due to the timing constraints. Downsizing gate I results in a circuit with less power because both inputs of gate I are heavily toggled.

Cadence Synthesis Rapid Adoption Kit

More about Low Power Synthesis

Figure C-3 Gate Sizing

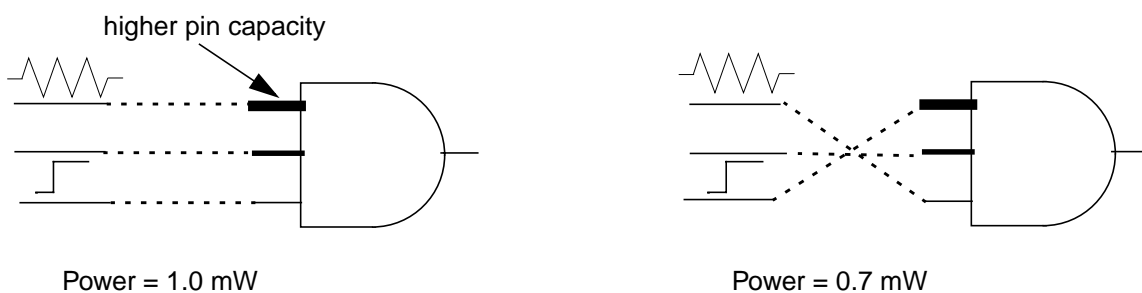


Pin Swapping

Pin swapping matches nets connected to logically equivalent pins so power dissipation can be minimized. LPS matches pins with higher capacitance to nets that have lower switching activity. Figure C-4 shows how pin swapping works.

Pin swapping can actually increase the delay of the design if not done carefully. LPS finds the optimal configuration that reduces power dissipation without affecting timing.

Figure C-4 Pin Swapping



Buffer Removal

Sometimes during timing optimization buffers are added to shield the critical path from a high capacitive load. However, during timing optimization, the critical path itself can shift resulting in unnecessary buffers on non-critical paths.

While these unnecessary buffers are removed in a postprocessing step, only a limited number can be removed without violating timing. To maximize power savings, the power optimizer removes unnecessary buffers that drive highly active nets.

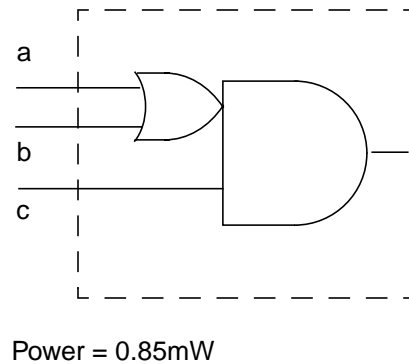
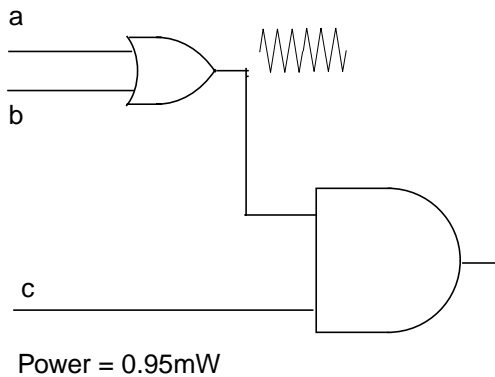
Note: In a normal timing optimization flow, regular buffer removal can increase power. The reason is that the power saved by removing a buffer can be significantly offset by the power increase at the buffer's fanout due to the increased slew. LPS performs a detailed analysis to ensure that the removal of each buffer indeed saves power.

Gate Merging

A major portion of dynamic power dissipation consists of driving the capacitive load of a net. A significant amount of dissipated power can be saved by dissolving the net because the dynamic power dissipation of a net is proportional to the switching activity of the net. These savings are achieved by merging the gates on either side of the net. See Figure C-5 to see an example of gate merging.

Note: Although gate merging eliminates the power loss on the net, it can increase the internal power dissipation of the new gate. LPS makes the correct tradeoffs.

Figure C-5 Gate Merging

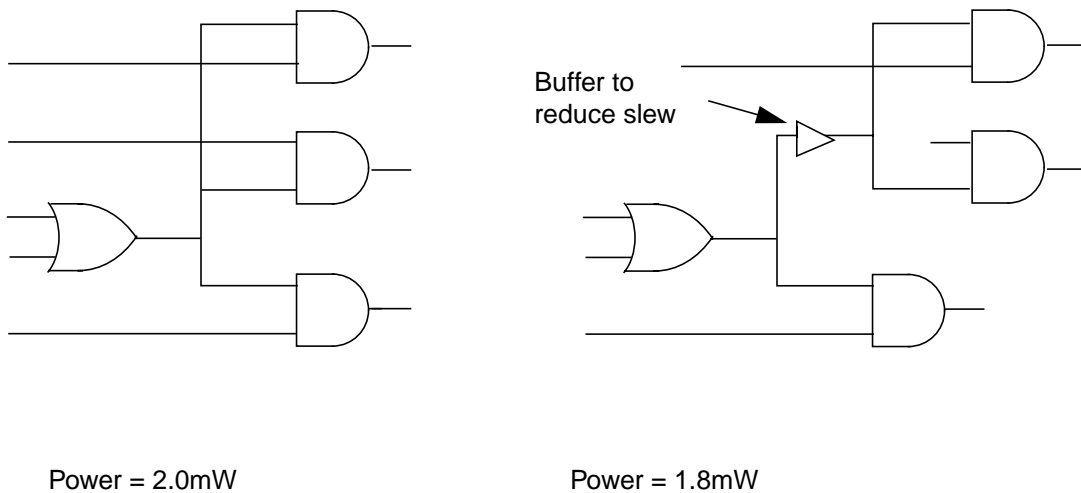


Slew Optimization

The internal power of a cell largely depends on the slew of the input signals. The larger the input slew, the larger the power dissipated by this gate. If a signal has a large slew, driving the signal through a buffer improves the signal's slew and reduces the overall power of the cells driven by the signal.

Figure C-6 shows an example of how slew optimization works during timing-constrained power optimization. LPS reduces slew by adding buffers. The challenge for LPS is to choose the set of gates that it needs to buffer. LPS performs a detailed analysis to identify the optimal partition that will reduce power as much as possible without violating timing and adds a buffer of optimal size to that cluster.

Figure C-6 Slew Optimization



Logic Restructuring

Logic restructuring is different from other transformations performed during power optimization. Unlike gate merging, buffer removal, and other small local transformations that do not make significant changes to the circuit structure, logic restructuring actually changes the topological structure and internal functions of a circuit without compromising the circuit's functional integrity.

Figure C-7 on page 179 shows a circuit before logic restructuring. Signal A has the lowest switching activity among all inputs. Although the final output of the circuit has a low switching activity, the two internal nodes are highly active, causing the bulk of switching power dissipation.

Figure C-7 Circuit Before Logic Restructuring

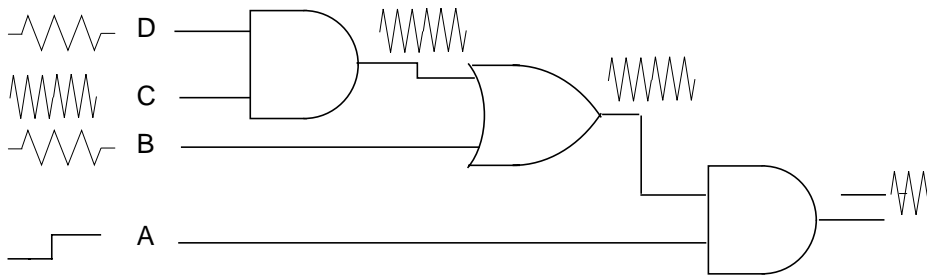
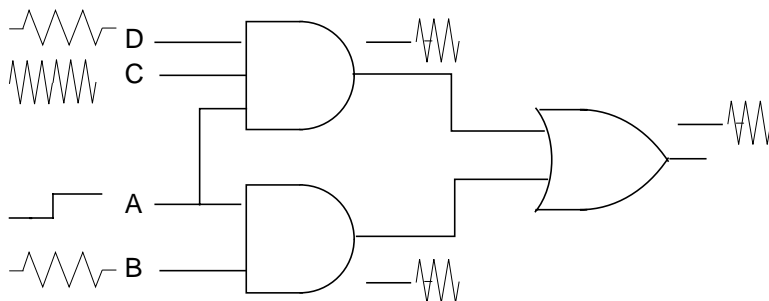


Figure C-8 on page 179 shows how logic restructuring can find a more efficient circuit implementation that cannot be done by other power optimization transformations. The new circuit dissipates much less power than the original circuit shown in Figure 3-9 because the logic functions at the two internal nodes are now less active.

Figure C-8 Circuit After Logic Restructuring



Cadence Synthesis Rapid Adoption Kit
More about Low Power Synthesis

Return to [Analyzing the Results and Generating Reports](#) on page 137.