PowerMill User Guide

Release 5.4, January 2000

Comments?
E-mail your comments about Synopsys documentation to doc@synopsys.com

SYNOPSYS®

Document Order Number: 32647-014 HB
PowerMill User Guide, Release 5.4

PowerMill User Guide

Table of Contents

About This Manual

The *PowerMill User Guide* describes how to use the PowerMill simulator to do power simulations on and analysis of integrated circuit designs.

For information on:	See:
A general overview of the features and capabilities of the PowerMill simulator	Chapter 1, “Introduction to PowerMill”
The basic steps for preparing initial files, running a simulation, and viewing the results of a PowerMill simulation	Chapter 2, “Getting Started”
Tutorials that give detailed instructions and scenarios of how to use PowerMill	Chapter 3, “PowerMill Tutorials”
Procedures for doing different types of power analyses	Chapter 4, “Power Analysis”
General information on using the ACE feature to simulate analog and mixed-signal designs	Chapter 5, “Using the ACE Feature”

For information on:	See:
Tutorials that give detailed instructions and scenarios of how to use the Analog Circuit Engine (ACE) option of the PowerMill simulator	Chapter 6, "ACE Tutorials"
Instructions for using the PowerMill Graphical Analyst	Chapter 7, "Using the PowerMill Graphical Analyst"
A tutorial on the PowerMill Graphical Analyst	Chapter 8, "PowerMill Graphical Analyst Tutorial"
Sample output files	Appendix A, "Sample Power Reports"
Using shared memory	Appendix B, "Using Shared Memory"

Audience

This manual is for integrated circuit engineers and designers performing power simulations to test and verify integrated circuit designs. Knowledge of UNIX, high-level design techniques, and circuit description tools, such as SPICE, is assumed.

Related Manuals

The following Synopsys manuals in the EPIC tool set provide additional information on using PowerMill:

- *PowerMill Reference Guide*
- *EPIC Tools Reference Guide*
- *ADFMI Manual*

Conventions

This manual uses certain style conventions to indicate commands, menus, examples, and filenames.

Commands

SYNTAX:

command_name [*argument(s)*]

argument types: keyword | *value* | tag=*value* | tag=keyword

Command Argument	Definition
keyword	Keywords are identifiers that must be used as they appear. They are shown in base font.
value	Values are user-determined. They are shown in italic text to distinguish them from commands and keywords.
tag= *value* *keyword*	Tags can be followed by either a value or a keyword. Tags and keywords are in the base font. Argument values are in italics to distinguish them from commands, keywords, and tags.

Symbol	Definition
|	A pipe symbol (|) represents the word “or” and separates choices between two or more arguments.
...	An ellipsis (...) indicates that more than one argument can be specified. Ellipses are used only for multiple arguments with tags.

Symbol	Definition
[ ]	Open and closed square brackets indicate that the enclosed arguments are optional.
( )	Open and closed parenthesis indicate that there is a choice between the enclosed arguments (two or more). These are used only when a command has several groups of argument choices; multiple pipe symbols (|), in this case, would result in ambiguous syntax.

SYNTAX:

report_node_i a[vg] | r[ms] | p[eak] | h[ist] *node_name(s)*

For this command, a[vg], r[ms], p[eak], and h[ist] are keywords–choose one of these. The *node_name(s)* are user-determined.

EXAMPLE 1:

```
report_node_i a VDD GND
```

In this example, `a` is a keyword, and `VDD` and `GND` are values.

SYNTAX:

report_node_ic [a[ll]] [q[uoted]] [for=epic | spice] [*time(s)*]

For this command, a[ll] and q[uoted] are optional keywords. The tag for= is part of an optional argument for which you can choose the keyword epic or spice. The *time(s)* are user-determined and, in this case, optional.

EXAMPLE 2:

```
report_node_ic all for=spice 1u
```

In this example, `all` is a keyword, `for=` is a tag, `spice` is a keyword, and `1u` is a value.

Menu Text, Filenames, and Examples

Menu text appears in bold, as shown in the following example.

EXAMPLE 1:

To start setting up a run, select **File→Design Data Setup**.

Filenames are shown in the same font as the surrounding text, but in italics.

EXAMPLE 2:

This is an example of the file *filename.out* being shown in text.

Examples are shown in courier font as they might appear on your screen. Sometimes explanations follow examples, and in such cases, anything shown in the example that appears in the explanation is shown in the same courier font used in the example.

EXAMPLE 3:

```
add_node_cap RS100 275
```

In the example, the capacitance value of node `RS100` is increased by `275` femtofarads.

Chapter 1

Introduction to PowerMill

Overview

PowerMill is a transistor-level power simulator and analyzer for CMOS and BiCMOS circuit designs. As more components are integrated into smaller silicon chips, power consumption becomes a major concern. Detailed current information about each circuit component is critically important to the designer. Designs must be optimized for minimum power consumption.

PowerMill's effective power and current simulation and diagnosis capabilities not only enable you to successfully engineer the design of an integrated circuit, but also significantly improve its reliability.

Major Functions and Features

The PowerMill simulator can perform the following functions:

- Run transistor-level simulations (with SPICE-like accuracy) with a run time 10 to 1000 times faster than SPICE.
- Display instantaneous current waveforms.
- Display first derivative di/dt waveforms.
- Report peak, average, and RMS currents.
- Report DC-leakage paths.
- Perform static power diagnosis.
- Provide block power statistics that identify hot-spots—blocks with excessive power dissipation.
- Run in batch mode with different parameters.
- Automatic vector generation for maximum power estimation.

PowerMill also provides the following features:

- An Analog Circuit Engine (ACE) feature that allows you to perform accurate SPICE-like circuit simulation on much larger designs than those traditionally handled by SPICE. This feature supports bipolar transistors.

- An algorithm to efficiently handle node-to-node coupling capacitance and crosstalk.
- Two graphical analysts (one for the base simulator and the other for ACE) for setting up simulation runs and analyzing results.
- Interactive debugging tool.
- Technology migration requiring limited technology-related data.
- Automatic technology file generation.
- Standard MKS engineering units for all data. Values for time, voltage, current, capacitance, resistance, and inductance are represented as seconds, volts, amps, farads, ohms, and henries, respectively.

In addition, PowerMill supports these modeling features:

- The use of multi-level models, comprised of behavioral, gate, switch, and transistor models.
- The Analog Digital Functional Model Interface (ADFMI) for writing functional models using the C programming language (see the *ADFMI Manual*).

Inputs, Outputs, and Interfaces

PowerMill's inputs are a netlist, configuration file, technology file, and stimulus. See Chapter 2, "Getting Started" for more information on input files.

PowerMill's outputs can be textual or graphical. The SimWave tool from System Science, the turboWave from E-Team Design Systems, the waveform tools in Cadence Design Framework II, or VIEWlogic PowerView can be used to view analog current and voltage waveforms.

A PowerMill interface is available in Cadence (Opus and Edge), VIEWlogic, System Science in addition to the PowerMill Graphical Analyst and the ACE Graphical Analyst.

PowerMill also reports the peak, average, di/dt, and RMS current statistics of individual components, subcircuit blocks, and global power supply pads.

Chapter 2

Getting Started

Overview

This chapter provides the basic information you need to start using the PowerMill simulator. It lists the required and optional input files and provides a basic procedure for running a PowerMill simulation.

You can learn to run both basic and advanced PowerMill simulations by performing the tutorials provided in Chapter 3, "PowerMill Tutorials."

The PowerMill Environment

Figure 1 graphically represents the PowerMill environment.

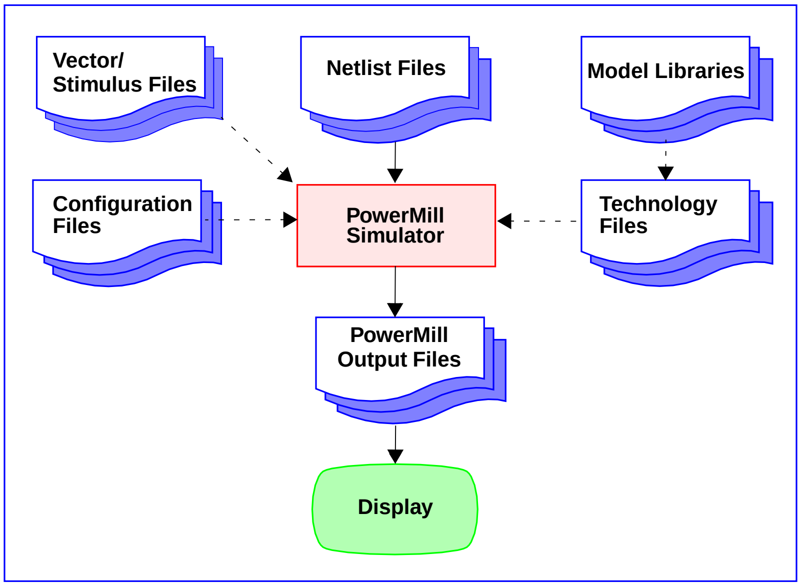


Figure 1 The PowerMill Environment

In this figure, solid arrows indicate the necessary input files for PowerMill. Arrows with dashed lines indicate optional files. Netlist files, technology files, and stimulus functions are fully described in the *EPIC Tools Reference Guide*.

The PowerMill Command

PowerMill is executed by typing the **powrmill** command followed by a series of command-line options. The -n command-line option is the only required option. The -p (to specify the technology file) and -c (to specify a configuration file) options are optional, but frequently used.

EXAMPLE:

```
powrmill -n netlist.spi -c cfg -o test -p typ_tech*
```

If no configuration file is specified and a SPICE or HSPICE netlist is used, the **use_sim_case** configuration command with the *l* (lowercase letter L) argument specified, will be set as the default. This is because HSPICE is case-insensitive and sets all characters to lowercase.

See Chapter 1, "The PowerMill Command" in the *PowerMill Reference Guide* for details on all of the PowerMill command-line options.

Batch Versus Interactive Mode

PowerMill can be run in batch mode or interactive mode. The default is batch mode. In this mode, PowerMill runs to completion without intervention. To use PowerMill interactively, you must specify the -i option or type Control-c while the simulation is running (that is, after netlist compilation). The interactive commands are described in Chapter 2, "Configuration and Interactive Commands" in the *PowerMill Reference Guide*.

Environment File

PowerMill lets you define certain simulation environment parameters in the *.epicrc* file. The PowerMill simulator uses these parameters, but they can be overridden with options on the command line. The syntax, description, and example of this file are included in the section "Creating the Environment (.epicrc) File" on page 13.

Netlist Files

The netlist file describes the system or circuit to be simulated, and is required to run a simulation. The following formats are accepted by PowerMill: EPIC, HSPICE/SPICE, EDIF, LSIM, Verilog, and Cadence SPF. More than one netlist, with the same or different formats, can be specified simultaneously; for example, you can combine SPICE, EPIC, Verilog, and EDIF in the same run. If a format is not specified, PowerMill uses the file extension to determine the format. For information on supported netlist formats and their corresponding extensions, see Chapter 3, "Netlist Compiler and Translators" in the *EPIC Tools Reference Guide*.

Using HSPICE/SPICE Format

If a netlist file with a *.sp* or *.spi* extension is used, the parser assumes the netlist is in an HSPICE format and sets all nodes and elements to lowercase. Don't use case-sensitive node names and element names if you are going to use SPICE netlists.

If a SPICE netlist is not specified, the netlist input, by default, is case sensitive. You can override this default setting with the **use_sim_case** configuration command.

HSPICE Netlist Compatibility

PowerMill is capable of recognizing .measure commands in HSPICE netlists. For more information about HSPICE netlist commands, see"HSPICE Netlist Syntax" in Chapter 3 of the *EPIC Tools Reference Guide*.

Using Other Formats

For information on supported netlist formats, see Chapter 3, “Netlist Compiler and Translators” in the *EPIC Tools Reference Guide*.

Configuration Files

The configuration file contains information that tells PowerMill how to perform the simulation. Any number of configuration files, including none, can be used. The configuration file contains run parameters data:

- Nodes to display during simulation (analog or digital values).
- Power analysis to be performed.
- Circuit modification.
- Simulation accuracy control.

The configuration commands are fully documented in Chapter 2, “Configuration and Interactive Commands” and Chapter 3, “ACE Configuration Commands” in the *PowerMill Reference Guide*.

Technology Files

The technology file describes the key features of the process technology that PowerMill uses to predict the transistor behavior in the circuit. This technology file is created using a utility called Gentech. It can be created automatically or manually.

For information on how to automatically create a technology file see the “Automatic Technology File Generation” section in Chapter 4 of the *EPIC Tools Reference Guide*.

For more information on manually running the Gentech utility, see the “Gentech” section in Chapter 4 of the *EPIC Tools Reference Guide*.

Directly Supported Models

If you are using BSIM1, BSIM2, BSIM3 v.3, MOS9 (HSPICE level 50), MESFET, or JFET models, you don't need to specify a technology file. The technology data lookup tables are created internally at run time.

Stimulus Files

Stimulus information can be placed inside the netlist or included on the command line as a separate file. Types of stimulus include SPICE piecewise linear (PWL) or PULSE waveforms, EPIC clk or tgl patterns, and digital vector files.

To learn more about stimulus types, see Chapter 5, "Stimuli and State Checking" in the *EPIC Tools Reference Guide*.

Creating the Environment (.epicrc) File

A file named *.epicrc* can be used to specify most of the command-line options for the **powrmill** command. Although not required, it is recommended for specifying options that are used frequently. For detailed information on PowerMill command-line options, see Chapter 1, "The PowerMill Command" in the *PowerMill Reference Guide*.

PowerMill searches for the *.epicrc* file in three directories in the following order:

1. Your current working directory.
2. Your home directory.
3. The EPIC installation directory, as defined by $EPIC_HOME.

If there are multiple *.epicrc* files in these directories and/or command-line options to control the settings, the precedence in descending order is:

1 Command-line option settings

2 The *.epicrc* file settings in your current working directory

3 The *.epicrc* file settings in your home directory

4 The *.epicrc* file settings in $EPIC_HOME

CAUTION: If a file named *.epicrc* already exists in your home directory for a different simulator, a warning message is printed when PowerMill is run.

The following table lists PowerMill command-line options and their corresponding keywords in the *.epicrc* file.

Command-Line Option	Corresponding Keyword in the .epicrc File
-A	`analog_mode`
-c	`config_files` `default_config_files`
-d	`delay_mode`
-F	`no_ace`
-fm	`user_adfm_obj_modules`
-FM	`user_adfm_obj_modules_force`
-L	`cell_lib_path`
-m	`top_cell`
-n	`netlist_files`
-o	`output_pefix_name`
-out	`print_format`
-p	`technology`
-t	`running_time`
-u	`user_obj_modules`

Command-Line Option	Corresponding Keyword in the .epicrc File
-U	`user_obj_modules_force`
	`user_libraries`

NOTE: The -c command-line option will not override the `default_config_files` keyword, instead, they are concatenated.

NOTE: You can use the `user_libraries` keyword, which does not have a corresponding command-line option, to specify the full path to the *libFuncModel* library, used with built-in ADFMI models. See Chapter 7, “Built-in Models” in the *5.4 ADFMI Manual* for more information on the *libFuncModel* library.

Sample .epicrc File

```
; This is a sample .epicrc file.
powrmill:netlist_files:dram.net
powrmill:config_files:cfg
powrmill:running_time:1000
powrmill:output_root_name:dram
powrmill:technology:techfile
powrmill:user_adfm_obj_modules:dram.o
powrmill:analog_mode
```

The semicolon (;) begins a comment line in the *.epicrc* file.

Running a Basic Simulation

This section describes the procedure for running a basic PowerMill simulation in batch mode. For information on running a simulation using the Graphical Analyst, see Chapter 7, “Using the PowerMill Graphical Analyst.”

1 Set up your environment so you can access the directory that contains the current PowerMill release.

2 Have the following input files in your current directory:

- A *.epicrc* file (optional)
- Netlist files in any supported format including, SPICE (xxx.*spi*), HSPICE (xxx.*spi*), and EPIC format (xxx.*ntl*)
- Technology files (optional)
- Configuration files (optional)
- Stimulus files (optional)

3. Create a run script for your simulation. This script should contain the **powrmill** command with the appropriate options specifying your input files.

   EXAMPLE:
   ```
   powrmill -n adder.spi -c cfg -p techfile -o adder
   ```

4. Give the script a meaningful name such as *runpw* or *runpw.scr*.

5. Run the simulation by entering the name of the run script.

   EXAMPLE:
   ```
   runpw
   ```

   These output files will be generated: *adder.log*, *adder.err*, and *adder.out*.

   For more information on output files, see Chapter 4, “PowerMill Output Files” in the *PowerMill Reference Guide*.

6. View the output files with the turboWave or SimWave waveform viewer. To do so, type in the name of the viewer followed by the name of the output file you want to view.

   EXAMPLE:
   ```
   turboWave -f adder.out
   ```

Chapter 3

PowerMill Tutorials

Overview

This tutorial demonstrates both basic and advanced PowerMill simulations using the 4-bit adder shown in Figure 1.

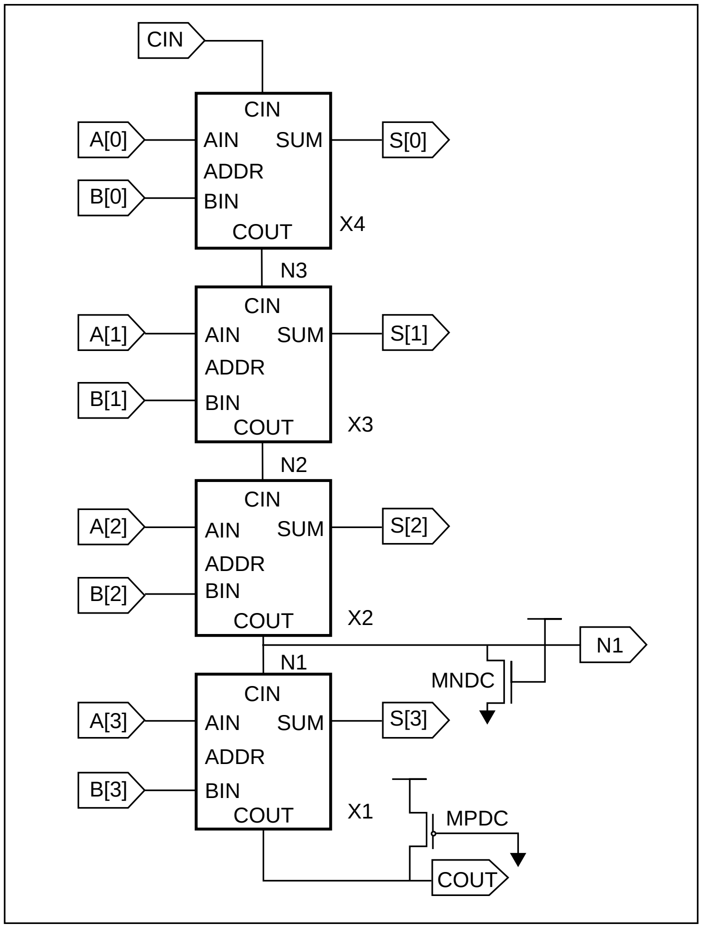


Figure 1 Logic diagram of the four-bit adder circuit

This circuit contains 164 transistors and is simulated with 150 vectors. Figure 2 and Figure 3 further illustrate the adder circuit used in this tutorial.

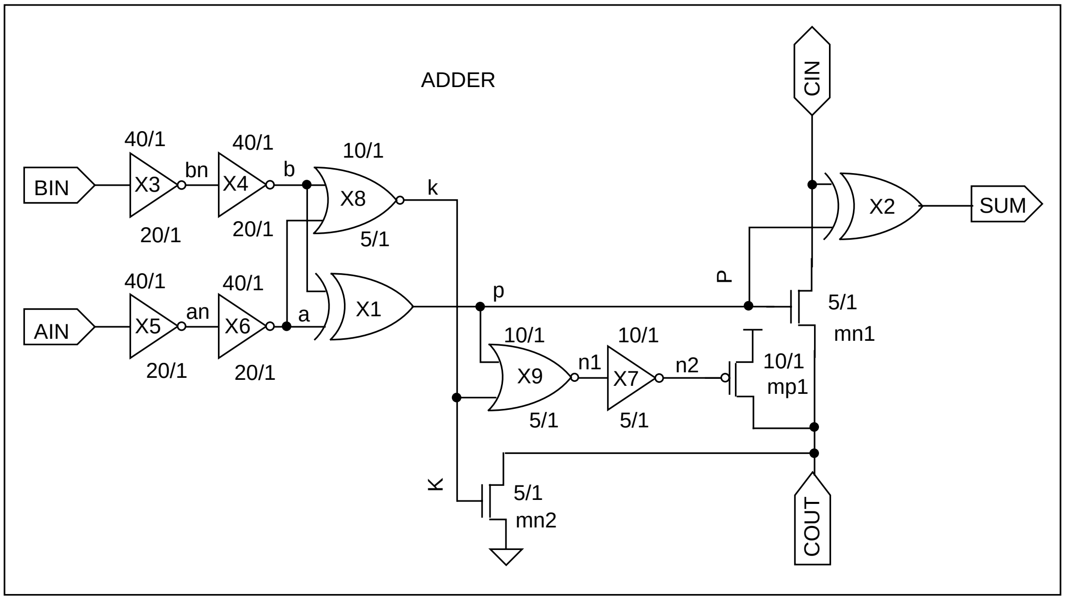


Figure 2 Logic diagram of the adder cell

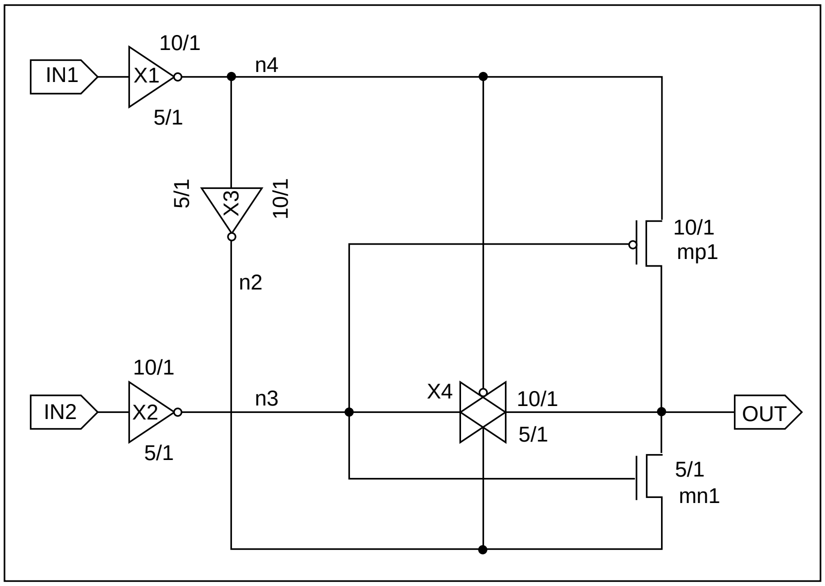


Figure 3 Logic diagram of an XOR2 cell

Tutorials Included in this Chapter

This chapter includes the following tutorials:

- Tutorial 1: Running a Basic Simulation
- Tutorial 2: Performing a Full-Chip Power Analysis
- Tutorial 3: Performing a Block-Level Power Analysis
- Tutorial 4: Finding DC Paths
- Tutorial 5: Estimating Maximum Power for Combinational Circuits
- Tutorial 6: Estimating Maximum Power for Sequential Circuits
- Tutorial 7: Customizing a GAP Objective Function using an ADFMI Code File
- Tutorial 8: Finding Static Leakage Paths

NOTE: The turboWave screens in this chapter are displayed with altered colors and signal height to improve readability. Therefore, the waveforms you actually see in turboWave will look significantly different from the screens in this chapter.

Getting the Input Files

Before you can run any of the tutorials in this chapter, you need to copy the required input files to your current working directory. The following procedure shows you how to do this.

1 Set the correct path for the EPIC_HOME environment variable. If $EPIC_HOME is not set, see your system administrator.

2 Copy, recursively, the files from *$EPIC_HOME/tutorials/pw_basic* and *pw_advanced* to your local working directory. The files in these directories (and their subdirectories) are needed for the tutorials in this chapter.

```
cp -R $EPIC_HOME/tutorials/pw_basic .
cp -R $EPIC_HOME/tutorials/pw_advanced .
```

These commands copy the *pw_basic* and *pw_advanced* directories, respectively, into the current directory (the period tells the **cp** command to copy to the current directory).

3. Verify that your newly copied *pw_advanced* directory contains the following subdirectories: *block_powr*, *dc_path* *full_chip*, *max_powr1*, and *max_powr2*. Each of these directories corresponds to and contains the files needed for an advanced tutorial in this chapter.

SPICE Netlists

All tutorials in this chapter use SPICE format netlists. Conversion to EPIC format is not necessary.

A Basic PowerMill Simulation

The tutorial in this section demonstrates a basic PowerMill simulation using the UNIX batch mode.

Tutorial 1: Running a Basic Simulation

The following table lists the files needed for this tutorial. These files are located in the *pw_basic* directory.

Filename	Description
cfg	Batch run configuration file
adder.spi	SPICE netlist for adder circuit
adder.vcd	Verilog description file
adder.vtran	Control file for changing Verilog into EPIC format
run	Run script for this tutorial
tech.typ.25c_5v	Technology file

Procedure

Use the following procedure to run the basic tutorial.

1 Use the UNIX **cat** command to display the contents of the run script.

```
cat run
```

This script contains the following line:

```
powrmill -n adder.spi adder.cmd -c cfg -o adder
   -p tech*
```

The run script specifies the SPICE netlist, `adder.spi` and the command file for processing the netlist, `adder.cmd`. It also specifies the `cfg` and `tech*` configuration and technology files. In addition, the -o option setting instructs PowerMill to use a prefix of `adder` for all output files resulting from this run.

See Chapter 1, "The PowerMill Command" in the *PowerMill Reference Guide* for more information on command-line options.

NOTE: There is no "e" in the **powrmill** command.

2 Use the UNIX **cat** command to display the contents of the configuration file.

```
cat cfg
```

This file contains the following configuration commands:

```
print_node_logic *

print_node_v *

report_node_powr gnd vdd
```

The first command prints all nodes in a digital format. The second command prints all nodes as voltage waveforms. The last command creates a report, in the *adder.log* file, of the average, RMS, and the five highest peak currents in the circuit. Average and RMS waveforms are also be printed to the *adder.out* file.

See Chapter 2, “Configuration and Interactive Commands” in the *PowerMill Reference Guide* for more information on these commands.

3 Use the **VTRAN** program to create the vector stimulus file for this simulation.

```
vtran adder.vtran
```

Messages indicating the execution of the **vtran** command are displayed. **VTRAN** generates two files: *adder.cmd* and *adder.vec*. The *adder.vec* file contains the tabular vectors for this simulation.

The *adder.cmd* file contains the following line:

```
(is=vec) (en=adder.vec) (ot=cin,a[3-0],b[3-0]);
```

This line calls the *adder.vec* file and provides the signal order, for the vector file, to the simulator.

See the *VTRAN 3.5 User Manual* for more information.

4 Run the simulation with the *run* script.

5 When the simulation is finished, list the directory contents.

Several output files with the prefix *adder* are created.

6 Use turboWave to view the waveforms.

```
turboWave &
```

7 From the turboWave main window, select **File→Open** to open the *adder.out* file.

8 Use the **Signal** pull-down menu to select the signals to view (see Figure 4).

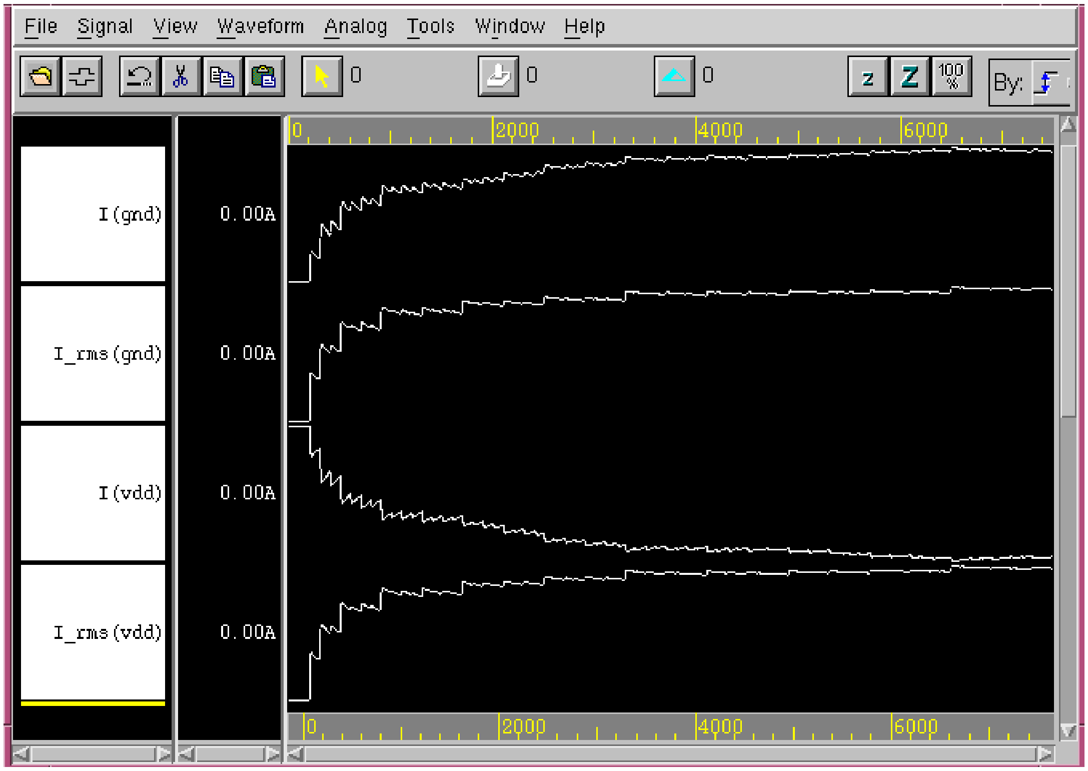


Figure 4 Waveform of basic adder circuit (selected signals)

9 When finished viewing the waveforms, select **File→Exit** to exit the program.

Advanced PowerMill Simulations

This section illustrates some of the power analyses you can do with the PowerMill simulator. Figure 5 shows the adder circuit used for these advanced simulations. This circuit is the same as that used in tutorial 1 except that here a 4-bit register has been added to the output.

Tutorials 2–6 illustrate the use of configuration commands related to different types and levels of power analyses. Once you understand how these commands work in separate simulations, you can combine them into a single run. Tutorials 2–4 cover only *digital* simulation.

Tutorials 2–6 are written with the assumption that you know how to run vi (or another text editor) and turboWave; therefore, the specific steps needed to open and reopen files, using these tools, are not described.

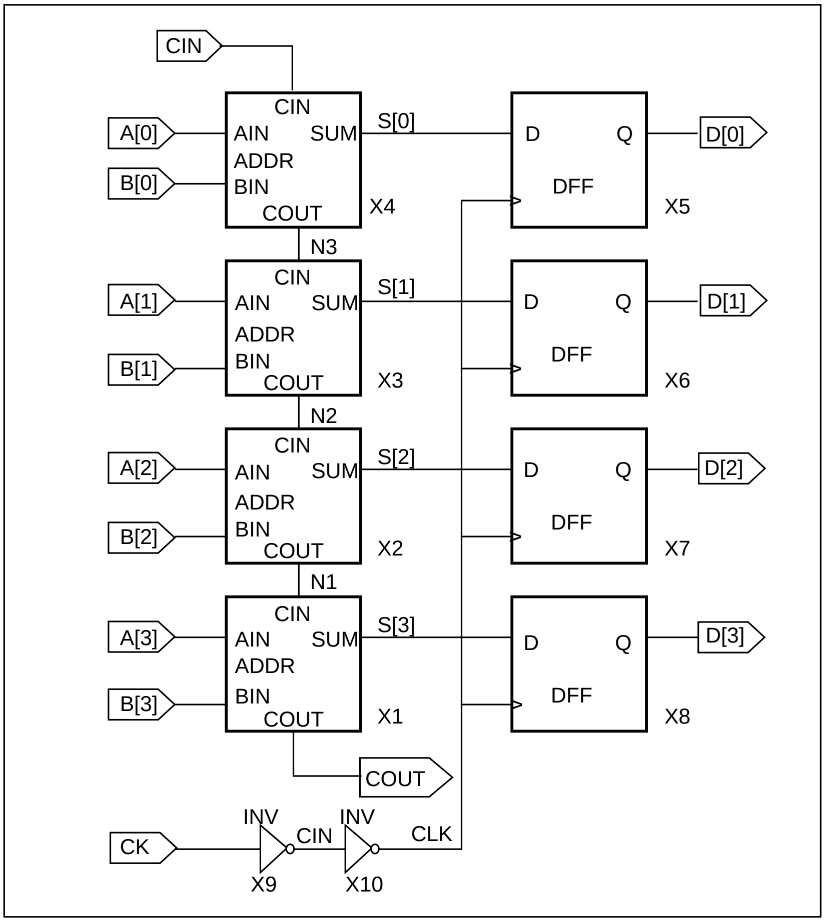


Figure 5 Logic diagram of the circuit for the advanced tutorials

Before You Begin

Before beginning any of the following tutorials, we recommend that you examine the run scripts and the configuration files for each tutorial in which you are interested.

1 Use the UNIX **cat** command to display the contents of each run script as needed.

```
cat full_chip/run_full
cat block_powr/run1_block
cat block_powr/run2_block
cat block_powr/run3_block
cat dc_path/run_dcpath
cat max_powr1/run_max_power
cat max_powr2/run_max_seq_power
cat max_powr3/run_custom
```

The preceding scripts contain the following command lines, respectively:

```
powrmill -n full.spi adder.cmd -c cfg_full
   -o full -p tech*

powrmill -n block.spi adder.cmd -c cfg1_block
   -o block1 -p tech*

powrmill -n block.spi adder.cmd -c cfg2_block
   -o block2 -p tech*

powrmill -n block.spi adder.cmd -c cfg3_block
   -o block3 -p tech*

powrmill -n dcpath.spi adder.cmd -c cfg_dcpath
   -o dcpath -p tech*

powrmill -n max_com_power.spi -c cfg_max_power
   -o maxpower -p tech*

powrmill -n max_seq_power.spi -c cfg_max_power
   -o maxpower -p tech*

powrmill -n max_seq_power.spi -c cfg_max_power -fm
gap_customize.c -o custom
```

All of the run scripts contain the **powrmill** command with the -n, -c, -p, and -o options specified.

See Chapter 1, “The PowerMill Command” in the *PowerMill Reference Guide* for more information on command-line options.

Although all of the simulations in this chapter use SPICE netlists, you can combine SPICE, EPIC, Verilog, and EDIF in the same run. If a netlist file with a *.sp* or *.spi* extension is used, the parser assumes it is an HSPICE netlist and sets all nodes and elements to lowercase by default.

2 Use the UNIX **cat** command to display the contents of the configuration files as needed.

```
cat cfg_full
cat cfg1_block
cat cfg2_block
cat cfg3_block
cat cfg_dcpath
cat cfg_max_power
```

Tutorial 2: Performing a Full-Chip Power Analysis

This tutorial shows you how to determine the power consumption of a complete circuit or full chip. In this case the “full chip” is a small adder circuit, with an output register. A small circuit is used in these tutorials so that simulations finish quickly.

Files Needed for this Tutorial

The following table lists the input files needed for this tutorial. These files are located in the *pw_advanced/full_chip* subdirectory in your working directory.

Filename	Description
adder.cmd	EPIC format netlist file that describes the filename for the vector file and the node names and port ordering for the vector file
adder.vec	Tabular digital vector file used as the digital stimulus for the simulation
cfg_full	Configuration file for full-chip power analysis
full.spi	SPICE netlist for full-chip power analysis

Filename	Description
run_full	Run script for full-chip power analysis
tech.typ.25c_5v	Technology file

Procedure

Use the following procedure to run the full-chip analysis tutorial.

1 Using the UNIX **cat** command, look at the contents of the *cfg_full* configuration file.

The following file sample shows the contents of this configuration file.

```
print_node_logic *
print_node_v *

;Power reports for average, capacitive, percent wasted
power
report_block_powr top track_gnd=1 track_wasted=1 *
;Print instantaneous, average, and rms waveforms
to.out file
print_probe_i inst top.*
print_probe_i avg top.*
print_probe_i rms top.*
```

The commands as set in this configuration file, perform the following functions:

- **print_node_logic**: prints all nodes in digital format.
- **print_node_v**: prints all nodes as voltage waveforms.
- **report_block_powr**: provides the averages for supply, ground, input, output, and biput as well as instantaneous waveforms in the *.out* file. It also provides a report on the percent of wasted power (short-circuit current).
- **print_probe_i**: prints the instantaneous, average, and RMS current waveforms.

2 Run the simulation using the *run_full* script.

3 Examine the *full.log* file.

Notice the reports at the end of the file.

4 Load the *full.out* file in turboWave and display the average power waveforms in each block, which are preceded by an upper-case I (see Figure 6).

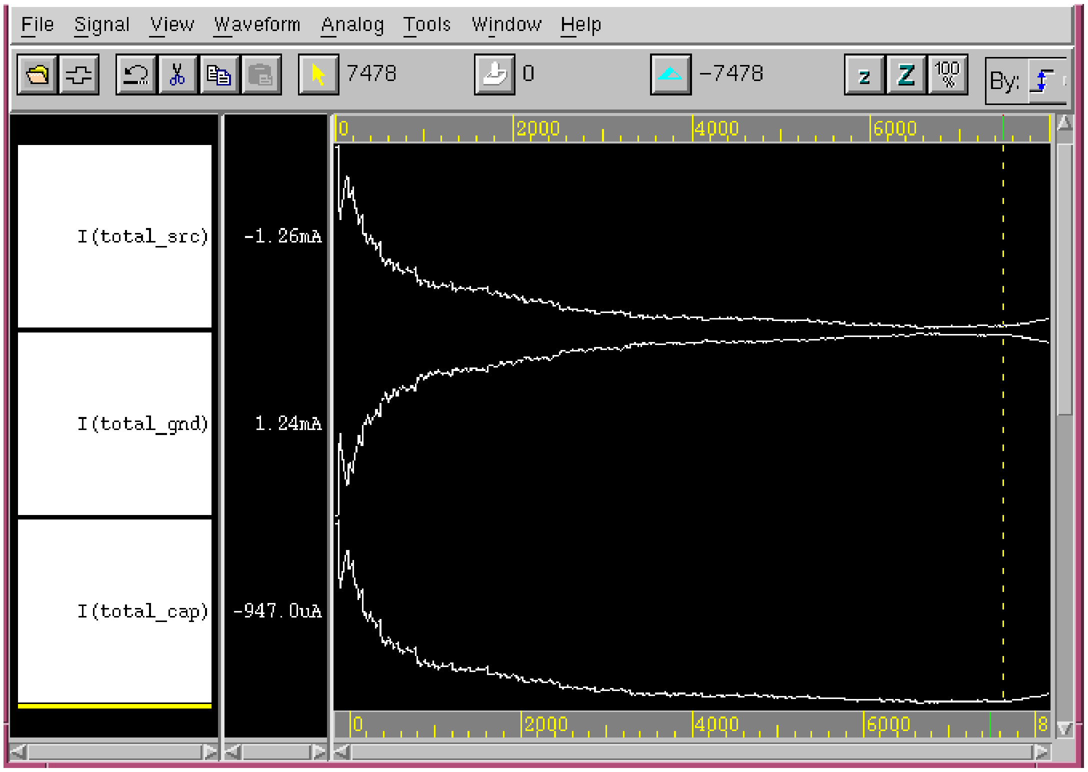


Figure 6 Waveform of advanced adder circuit (selected signals)

Tutorial 3: Performing a Block-Level Power Analysis

As part of the design process you probably need to determine how much power is consumed by each block. This information allows you to optimize blocks using the most power. PowerMill provides some configuration commands useful for retrieving information on power consumption at the block level. See "Analyzing Power Block-by-Block" on page 87 for details on the configuration commands for block analysis. This tutorial illustrates how you can use the PowerMill simulator to analyze power block-by-block.

Files Needed for this Tutorial

The following table lists the files needed for this tutorial. These files are located in the *pw_advanced/block_powr* subdirectory in your working directory.

Filename	Description
adder.cmd	EPIC format netlist file that describes the filename for the vector file and the node names and port ordering for the vector file
adder.vec	Tabular digital vector file used as the digital stimulus for the simulation
cfg1_block *cfg2_block* *cfg3_block*	Configuration files for block power analysis
block.spi	SPICE netlist of the circuit for block power analysis
run1_block *run2_block* *run3_block*	Run scripts for block power analysis
tech.typ.25c_5v	Technology file

Procedure

Use the following procedure to run the block-level power analysis tutorial.

1 Open each of the configuration files and study the commands used.

Each configuration file illustrates a different way to get block power information. The information returned is different for each file. Each command separates the VDD and GND current for each block by assigning unique probe names.

2 Run the simulation three times, once for each run script.

3 When the simulation is finished, list the directory contents.

   The directory now contains several output files with the `block1`, `block2`, and `block3` prefixes.

4 Open and examine the *block1.log*, *block2.log*, and *block3.log* files.

   Notice the differences between the block-level reports in the different *.log* files. In this case, only the current for each adder cell is reported; therefore, the blocks are small. The block power reporting principle applies to large blocks of a chip. See "Differences Between the Resulting Block Power Reports" on page 34 for a detailed analysis of each report.

5 Load the *block3.out* file in turboWave and display the average power waveforms for _gnd and _src in each block (see Figure 7).

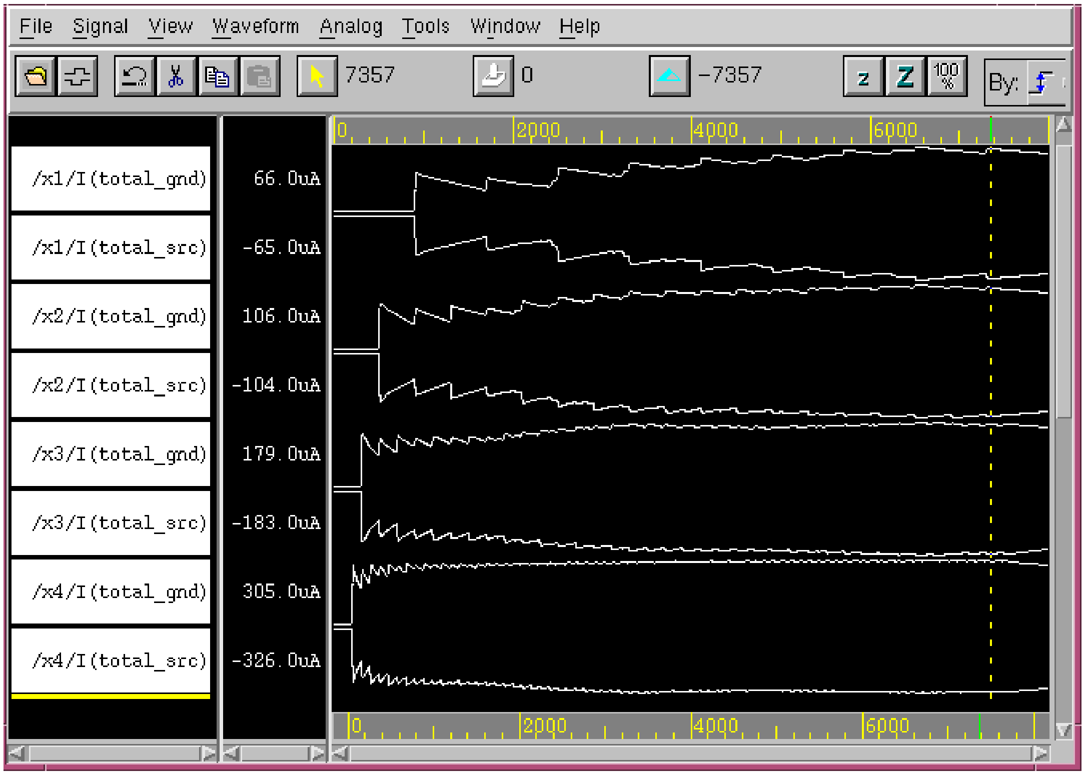


Figure 7 Waveforms of block analysis of output file block3.out (selected signals)

Differences Between the Resulting Block Power Reports

As this tutorial demonstrates, you can use the **report_block_powr** command with different options to produce reports specifically tailored to your needs. This section details the differences between the reports produced by the three block power runs in this tutorial.

block1.log

Figure 8 shows a portion of the block power reports printed in the *block1.log* file. These condensed reports provide information on the basic current consumption for each block. This information includes the average and RMS current for the block as well as the reported current peaks and the times at which they occurred. By default, the simulator provides the five highest peaks over the reporting interval. In this case, the reporting interval is the entire simulation. You can use the **set_print_ipeak** command to request the printing of additional current peaks.

```
Node: x1_vdd
    Average current     : -5.98305e+01 uA
    RMS current         :  3.13568e+02 uA

   Current peak #1    : -4.90200e+03 uA  at   2.50100e+02 ns
   Current peak #2    : -4.89100e+03 uA  at   4.10100e+02 ns
   Current peak #3    : -4.89000e+03 uA  at   5.70100e+02 ns
   Current peak #4    : -4.82500e+03 uA  at   9.01000e+01 ns
   Current peak #5    : -4.82500e+03 uA  at   7.30100e+02 ns

Node: x1_gnd
    Average current     :  6.10597e+01 uA
    RMS current         :  3.11328e+02 uA

   Current peak #1    :  4.90200e+03 uA  at   2.50100e+02 ns
   Current peak #2    :  4.89100e+03 uA  at   4.10100e+02 ns
   Current peak #3    :  4.89000e+03 uA  at   5.70100e+02 ns
   Current peak #4    :  4.82500e+03 uA  at   9.01000e+01 ns
   Current peak #5    :  4.82500e+03 uA  at   7.30100e+02 ns
```

Figure 8 Excerpt from block power reports in block1.log file

block2.log

Figure 9 shows a portion of the block power reports printed in the *block2.log* file. This particular excerpt provides a block power report for the `x1` block and was generated by the following command:

```
report_block_powr x1 track_src=1 track_gnd=1 x1.*
```

The report generated by this command lists the contents of and partitioning information for the reported block. This information consists of the number of stages used, the number of nodes in the block, and the total number of elements in the block.

This report also provides a detailed analysis of the current contributions from different parts of the circuit (inputs, outputs, and bidirectional ports, etc.).

```
Block: x1
    Number of nodes in block              :  18
    Number of elements in block           :  41
    Number of block supply nodes          :  1
    Number of block ground nodes          :  1
    Number of block biput nodes           :  1
    Number of block input nodes           :  2
    Number of block output nodes          :  0
    Number of block stages                :  10
    Number of block partial stages        :  1

    Average supply current                : -59.873924 uA
    RMS supply current                    :  313.624995 uA

    Average ground current                :  61.049494 uA
    RMS ground current                    :  311.347667 uA

    Average input current                 :  0.000000 uA
    RMS input current                     :  0.000000 uA

    Average output current                :  0.000000 uA
    RMS output current                    :  0.000000 uA

    Average biput current                 : -1.172278 uA
    RMS biput current                     :  30.454213 uA
```

Figure 9 Excerpt from block power reports in block2.log file

block3.log

Figure 10 shows a portion of the block power reports printed in the *block3.log* file. This excerpt shows the block power report for the `x1` block and was generated by the following command:

```
report_block_powr x1 track_wasted=1 track_power=1
   track_gnd=1 x1.*
```

This report is the same as that in the *block2.log* file with the addition of reports on capacitive contributions, wasted current, and power in watts.

```
Block: x1
    Number of nodes in block            :  18
    Number of elements in block         :  41
    Number of block supply nodes        :  1
    Number of block ground nodes        :  1
    Number of block biput nodes         :  1
    Number of block input nodes         :  2
    Number of block output nodes        :  0
    Number of block stages              :  10
    Number of block partial stages      :  1

    Average supply current              : -59.873924 uA
    RMS supply current                  :  313.624995 uA

    Average ground current              :  61.049494 uA
    RMS ground current                  :  311.347667 uA

    Average input current               :  0.000000 uA
    RMS input current                   :  0.000000 uA

    Average output current              :  0.000000 uA
    RMS output current                  :  0.000000 uA

    Average biput current               : -1.172278 uA
    RMS biput current                   :  30.454213 uA

    Average capacitive current          : -41.307089 uA
    RMS capacitive current              :  263.016049 uA

    Average wasted current              : -21.971519 uA
    RMS wasted current                  :  79.569210 uA

    Wasted current percentage           :  34.721875%

    Average block power                 :  302.205535 uW
    RMS block power                     :  1565.245971 uW
```

Figure 10 Excerpt from block power reports in block3.log file

Tutorial 4: Finding DC Paths

This tutorial illustrates how you can use the PowerMill simulator to find DC paths. The following table lists the files needed for this tutorial. These files are located in the *pw_advanced/dc_path* subdirectory in your working directory.

Filename	Description
adder.cmd	EPIC format netlist file that describes the filename for the vector file and the node names and port ordering for the vector file
adder.vec	Tabular digital vector file used as the digital stimulus for the simulation
cfg_dcpath	Configuration file for DC path checks
dcpath.spi	SPICE netlist of the circuit for DC path checks
run_dcpath	Run script for DC path checks
tech.typ.25c_5v	Technology file

Procedure

Use the following procedure to run the DC path check tutorial.

1 Open and examine the *cfg_dcpath* configuration file. The following file sample shows the contents of this configuration file.

```
print_node_logic *
print_node_v *
report_node_powr vdd gnd
report_ckt_dcpath
set_dcpath_thresh 200u
```

2 Run the simulation using the *run_dcpath* script.

Watch the on-screen messages as they scroll by during the simulation. These messages are included in the *.dcpath* file.

3 When the simulation is finished, list the directory contents.

The directory now contains several output files with the `dcpath` prefix.

4 Open at the *dcpath.dcpath* file and study the messages.

This file contains time stamps for when violations occur and lists the transistor at fault. Figure 11 shows the first DC path reported in the *dcpath.dcpath* file.

```
------------------------------------------------------------------
transistor x2.x2.x1.mn1        (type NMOS)  current = 345.30 uA
gate n2                                     voltage =   2.65 V
drain x2.x2.n4                              voltage =   0.61 V
source gnd                                  voltage =   0.00 V
transistor x2.x2.x1.mp1        (type PMOS) current =  -343.63 uA
gate n2                                     voltage =   2.65 V
drain x2.x2.n4                              voltage =   0.61 V
source vdd                                  voltage =   5.00 V
------------------------------------------------------------------
```

Figure 11 First DC path reported to dcpath.dcpath file

The **report_ckt_dcpath** command has other syntax options. This particular option with no arguments finds all DC paths that violate the 200 microamp threshold. In this case, the checks are triggered by any signal change.

5 Load the *dcpath.out* file in turboWave and display the cout, v(cout), n2, and v(n2) signals (see Figure 12).

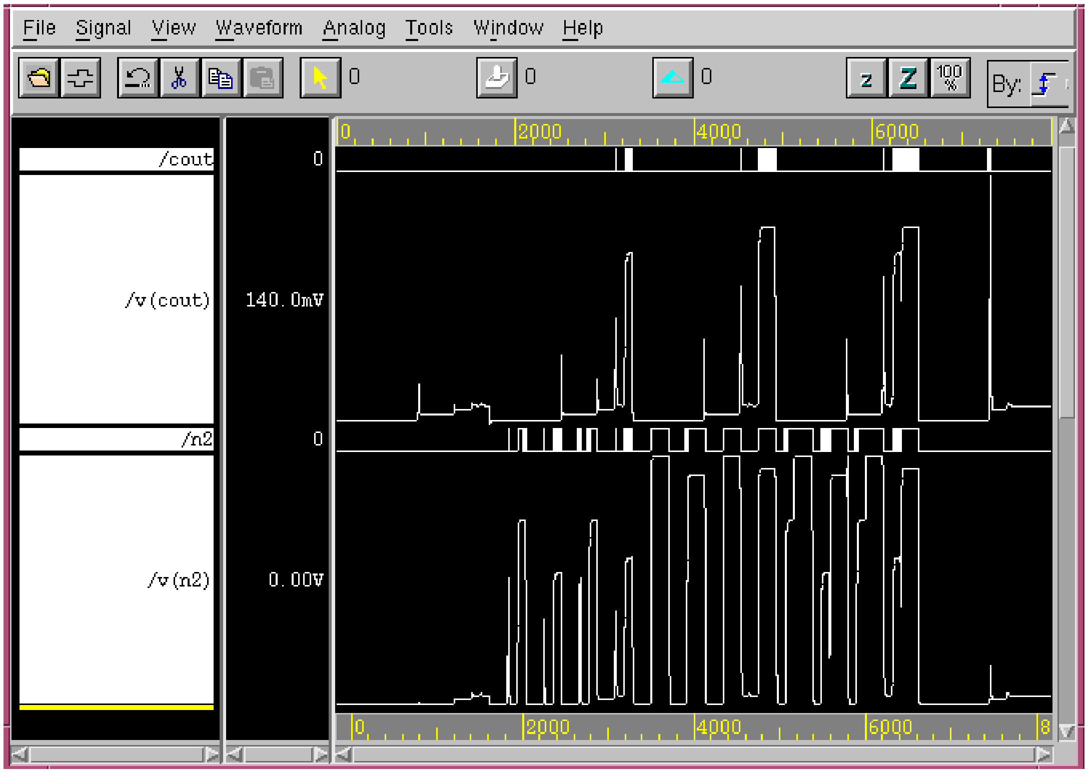


Figure 12 Waveform of DC path output file

Tutorial 5: Estimating Maximum Power for Combinational Circuits

This tutorial shows you how to estimate the maximum power of combinational circuits at the full-chip level. PowerMill's GAP (Genetic Algorithm for maximum Power estimation) feature combines a genetic algorithm with advanced circuit simulation techniques, to efficiently search for input vectors to maximize the power dissipation of a given circuit. You can use the GAP feature to generate a tight lower bound on the maximum power of a given circuit, which you can use to analyze various design issues (power management, for example.)

The GAP feature is able to maximize four objective functions:

- Average power per clock cycle of the circuit
- Instantaneous power of the circuit
- Average value per clock cycle of the customized fitness function
- Instantaneous value of the customized fitness function

The usage model for GAP customization, which involves the last two object functions, is illustrated in tutorial 7.

For combinational circuits, the maximum power (instantaneous or average) is produced by two vectors, referred to as a vector pair, applied in two consecutive clock cycles. The first vector sets up states of the internal nodes of the circuit. The power is measured in clock cycle II.

The GAP feature keeps a specific group of vector pairs as candidates for maximizing power dissipation. The quality of each vector pair is measured by the power consumption it produces. Using a genetic algorithm, the GAP feature repeatedly improves the quality of the vector pairs, in the group that was kept, generation by generation until the result is recognized by the simulator as satisfactory.

Files Needed for this Tutorial

The following table lists the files needed for this tutorial. These files are located in the *pw_advanced/max_powr1* subdirectory in your working directory.

Filename	Description
cfg_max_power	Configuration file for maximum power estimation
max_com_power.spi	SPICE netlist of the combinational circuit (a 4-bit ADDER) to be estimated
max_com_power1.vec	First vector file to be used for maximum power estimation
max_com_power2.vec	Second vector file to be used for maximum power estimation
run_max_power	Run script for maximum power estimation
tech.typ.25c_5v	Technology file

Procedure

Use the following procedure to run the tutorial on estimating maximum power for combinational circuits.

1 Open and examine the *cfg_max_power* configuration file. The following file sample shows the contents of this configuration file.

```
use_vec_gap vec=max_com_power1.vec vec=max_com_power2.vec
set_gap_opt period=20ns vec_history=1
guide_gap_search fit=1 level=5
print_probe_i inst total_power[gap]
```

The **use_vec_gap** command starts the GAP feature, and specifies two vector files with type vec to be used for the power estimation. The two vector files for the GAP estimation contain the following statements: signal names, type, radix (optional), and io (optional). Vector files for the GAP estimation differ from normal vector files in the following ways:

- ◆ A normal vector file must contain at least one vector. However, as you can see by examining the *max_com_power1.vec* and *max_com_power2.vec* files, a vector file for a GAP simulation can contain 0 vectors.
- ◆ Normal vector files have four valid io types: i (input), o (output), b (biput), and u (unused). Vector files for a GAP simulation have three valid io types: i (input), p (pseudo-primary input (PPI)), and b (biput). If no io statement is specified, all signals in the file are considered inputs.

Vector files for the GAP estimation can contain vector pairs to be used as part of the first generation for the genetic algorithm. All other first-generation vectors are randomly generated. You can use the **guide_gap_seach** command to specify the population size to control the number of input vector pairs applied in a generation.

The GAP estimation does not work on circuits in a wave-pipelining design; the clock period used in a GAP simulation must be longer than the delay along the critical path of the circuit. The default clock period is 20 ns. However, you can use the **set_gap_opt** command to override the default (with the period= option.) In the *cfg_max_power* configuration file, the **set_gap_opt** command specifies a clock period of 20 ns and a history option of 1 (the default value), therefore, only the vector pairs of the generation producing the maximum power are kept in the history files (*.gav* files). At the end of the estimation, GAP generates a history file corresponding to each vector file.

In this configuration file, the `fit=1` option (default) of **guide_gap_search** command instructs GAP to maximize the average power per clock cycle. You can choose to set up various stopping criteria for GAP using the **guide_gap_search** command (for example, CPU time, number of generations, and number of pairs). See the command definition for the **guide_gap_search** command in the *PowerMill Reference Guide*.

The `level=5` option is the default search level. Each search level (from 1 to 10) defines a corresponding parameter set used in the genetic algorithm. Selecting a larger search level generally produces a better (larger) estimate of the maximum power, but increases the CPU time.

2 Run the simulation using the *run_max_power* script.

3 Type `Control-c` to switch into the interactive mode.

In this mode, you can use the interactive GAP commands.

4 Type `report_gap_progress` at the interactive prompt.

This command shows the progress of the GAP estimation in a tabular format (default) to the screen. This table contains the maximum power, corresponding vector pair, and CPU time.

5 Type `report_gap_progress mode=2` at the interactive prompt.

This command displays GAP progress information, as a graph of the curve of the maximum power vs. the number of applied input vector pairs (see Figure 13).

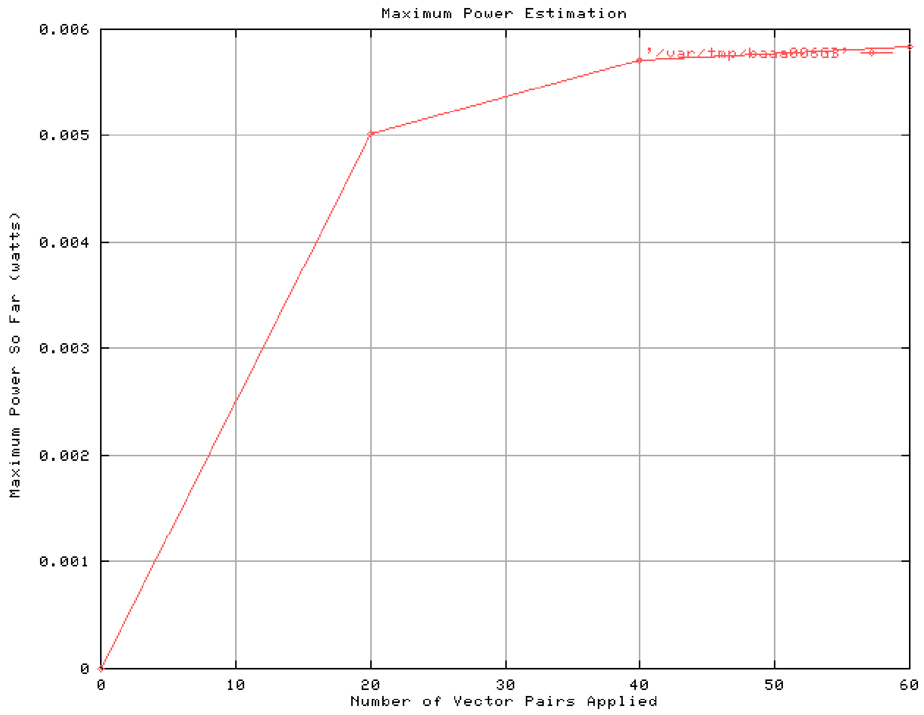


Figure 13 Graphical format of GAP estimation progress report

6 Close the window containing the graph.

7 Specify the `report_gap_parameters` command at the interactive prompt.

This command shows the current settings of various GAP options. In this tutorial, the following information is printed to the screen:

```
Current settings of parameters/options for GAP:

 Fitness function is peak average power
 History option                                   : 1
 Population size                                  : 20
 Search_level                                     : 5
 Clock period                                     : 20 (ns)
 No upper limit on CPU time
 No upper limit on number of generations
 No upper limit on number of vector pairs
 No upper limit on value of fitness function
```

8 Specify the `report_gap_prediction` command at the interactive prompt.

This command predicts a cut-off point for the estimation that you can use to reduce the total CPU time of the GAP estimation. The number returned is a prediction of the total number of vector pairs that need to be applied before the estimation converges. From this number, you can subtract the number of pairs already applied (as reported by the **report_gap_progress** command) to determine the number of additional pairs that need to be applied before the estimation converges.

For example, if the **report_gap_prediction** command returns 1000 and the **report_gap_progress** command reports that 700 vector pairs have already been applied, it is likely that the estimation will converge (produce the best result) after applying an additional 300 vector pairs. You can then apply the `guide_gap_search max_vec=300` command to cut-off the estimation appropriately.

NOTE: As with other extrapolation processes, the accuracy of the prediction from this command is limited. Although this command might reduce the CPU time for the simulation, it might also reduce the accuracy of the result. Therefore, you should only follow the prediction from **report_gap_prediction** command when reducing the CPU time is critical.

9 Specify `cont_sim` at the interactive prompt.

This restarts the GAP estimation.

10 When the estimate is done, list the contents of the current directory.

This directory contains output files with a *maxpower* prefix.

11 Open and examine the *maxpower.gap* file (see Figure 14).

```
Estimates of maximum average power from the GAP
feature:
---------------------------------------------------
(only those generations increasing maximum power are
listed)

 Generation 0
 Maximum power: 0.00540997 (watts)
 Vec[0]: 010000000
 Vec[1]: 101011011

 Generation 1
 Maximum power: 0.00559405 (watts)
 Vec[0]: 101011001
 Vec[1]: 110100101

 Generation 6
 Maximum power: 0.00640677 (watts)
 Vec[0]: 010010100
 Vec[1]: 101101001

 Generation 22
 Maximum power: 0.00653965 (watts)
 Vec[0]: 000000000
 Vec[1]: 110101101

 Generation 125
 Maximum power: 0.00653967 (watts)
 Vec[0]: 000000000
 Vec[1]: 110101101
---------------------------------------------------
Total number of generations: 152
Total CPU time             : 144.29 (secs)
```

Figure 14 Contents of the maxpower.gap file

This file contains a summary of the GAP estimation. The *maxpower.gap* file contains an entry for each generation that improves the maximum power. Each entry contains the following three fields: generation ID, maximum power induced, and the input vector pair producing the maximum power. The total CPU time and total number of generations are printed at the end of the file.

12 Open and examine the *max_com_power1.gav* and *max_com_power2.gav* files.

These are history files that save the input vectors used during the estimation. GAP produces a separate *.gav* file for each input vector file. The *max_com_power1.gav* and *max_com_power2.gav* files generated by this tutorial contain only the vectors of the generation producing the maximum power. You can specify which vectors you want GAP to keep using the vec_history= option of the **set_gap_opt** command.

See the *PowerMill Reference Guide* for details on all of the GAP configuration and interactive commands.

NOTE: You can use the *.gav* files (generated by GAP) as vector files for a regular PowerMill simulation (by specifying the -nvec command-line option). However, if you want to fully reproduce the GAP result for a specific generation, you have to consider the state of the circuit; therefore, you need to use the save and restore mechanism (described in tutorial 6).

13 Load the *maxpower.out* file in turboWave and view the mathematical signal gap.total_power. This signal represents the instantaneous power (in watts) of the combinational circuit used in this tutorial.

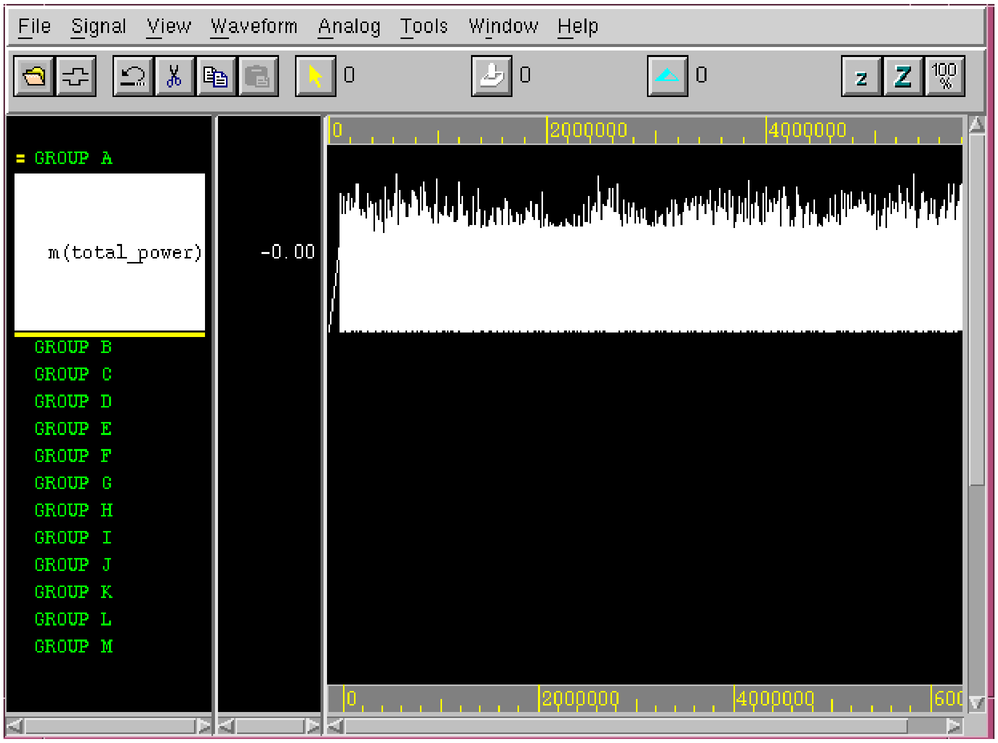


Figure 15 Waveform of m(gap.total_power) signal in maxpower.out file

Tutorial 6: Estimating Maximum Power for Sequential Circuits

This tutorial shows you how to estimate the maximum power of sequential circuits using PowerMill's GAP feature. This tutorial illustrates the following GAP features:

- Detection of pseudo-primary inputs (PPIs) using the **mark_node_latch** and **set_ckt_cmd** commands. Here PPIs are defined as the output of the memory elements (for example, latches and flip-flops).
- PPI detection using the PathMill latch detection feature.
- The save and restore feature using the save_history= option of the **set_vec_opt** command.

It is recommended that you first read the section, "Using the GAP Feature to Estimate Maximum Power" on page 103 in Chapter 3 before running this tutorial.

The GAP feature allows you to maximize four different objective (fitness) functions:

- Average power per clock cycle of the circuit
- Instantaneous power of the circuit
- Average value per clock cycle of the customized fitness function
- Instantaneous value of the customized fitness function

The usage model for GAP customization, which encompasses the last two bulleted items, is illustrated in tutorial 7.

For sequential circuits, such as the one shown in Figure 16, each measured power number is induced by three vectors, known as a vector triplet, that are applied in three consecutive clock cycles.

The first vector, which stimulates both primary inputs (PIs) and pseudo-primary inputs (PPIs), drives the circuit into a reachable state.The second and third vectors stimulate only PIs and leave all PPIs in a floating state. The power is measured in clock cycle III. This method allows both the primary input vectors and the state of the circuit to “evolve” through the use of a genetic algorithm to maximize the objective function.

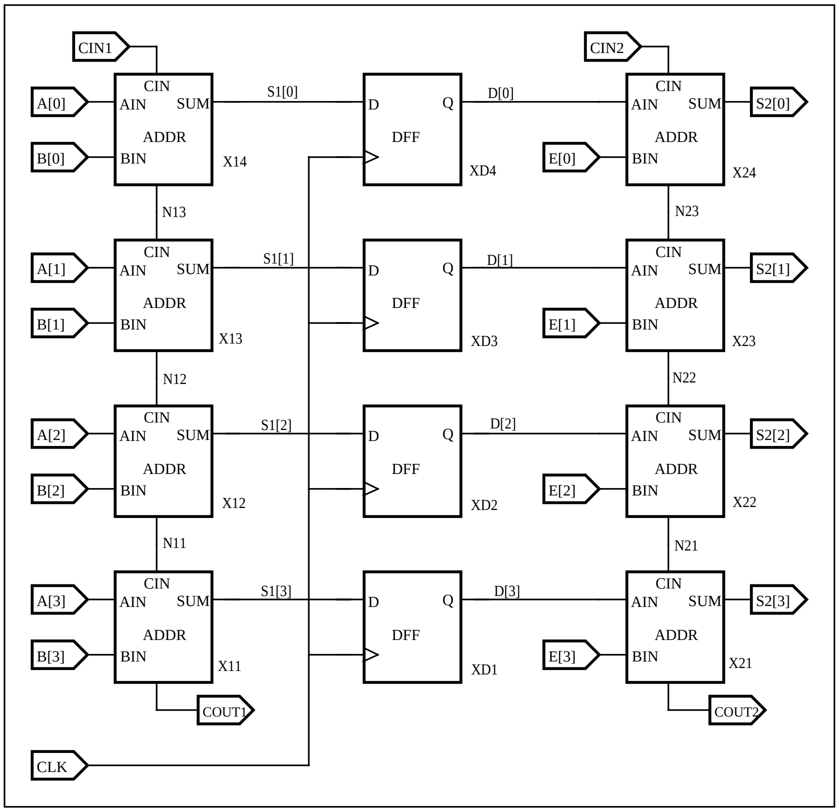


Figure 16 Schematic of the max_seq_power.spi netlist

For the vector-triplet method to work correctly on a sequential design, it must meet the following requirements:

- FFs (or latches) should be buffered at the output, so that the GAP stimuli on the PPIs won’t accidentally overwrite the content of the FFs.

  If the FFs are not buffered at the output, you need to verify some information about the circuit and perform an experimental GAP run to determine whether you should apply the vector-triplet method.

- FFs (latches) in the circuit should be universally positive edge-triggered (positive sensitive) or negative edge-triggered (negative sensitive).
- Clock skew in the estimated circuit (or block) should be optimized and negligible.

If the sequential circuit does not satisfy each of these requirements, the vector-triplet method might produce an unrealistic result. You can run the GAP feature on this type of circuit using the vector-pair method described in tutorial 5.

NOTE: The vector-pair method requires only PI information and estimates the design as a combinational circuit. The only drawback is that the quality of the result can be degraded, especially for large circuits, since there is no control over the state of the circuit during the GAP search.

Identifying the Pseudo-Primary Inputs

Before running the GAP feature using the vector-triplet method, the power estimation algorithm needs to identify the output of flip-flops and latches (the PPIs), so that they can be stimulated appropriately during the searching of optimal stimuli.

There are three different methods for identifying the PPIs: manual identification, automatic detection using PowerMill, detection using the PathMill simulator.

Manually Specifying the PPIs

You can manually specify the PPIs, along with PIs, in the GAP vector files. The vector files used for a GAP simulation, which use the same format as the EPIC vector files, provide information regarding the signals to be stimulated (for example, name, io, and radix).

This approach is only feasible for small sequential designs since identifying PPIs manually can be time consuming and error prone.

Using GAP's Built-in Mechanism

In hierarchical designs, FFs or latches might have subcircuit definitions. For these designs, you can use the **mark_node_latch** command, together with the **set_ckt_cmd** command, to identify their output automatically (see the command definition for **mark_node_latch** in the *PowerMill Reference Guide*). Similarly, you can use the **mark_node_latch** command and the **set_pattern_cmd** together to identify PPIs if FFs or latches are defined as patterns. Using this approach, you only need to provide GAP with vector files that include the PIs. GAP automatically generates and adds a vector file containing all the PPIs, identified by the **mark_node_latch** command, to the GAP estimation. This tutorial illustrates this method of PPI detection.

Using PathMill's Latch Detection Feature

If you have a circuit description that does not contain subcircuit or pattern definitions for flip-flops and latches, and you own a copy of the PathMill simulator, you can use PathMill on the circuit with the **pw_detect_ckt_latch** configuration command to detect latches. See "Using PathMill to Detect PPIs" on page 64 for details.

Testing Latch Functionality for Flip-Flops with Non-Buffered Outputs

If FFs (or latches) within the circuit are not buffered at the output, you have to perform an experimental GAP run to determine whether or not you can apply the vector-triplet method.

1 Use the **print_node_logic** command to print the logic waveform of the inputs and outputs of two or three FFs (for each type used in the circuit) for a few vector triplets.

2 For each triplet, verify that the values at the inputs by the end of clock cycle I are equal to the values at the outputs at the beginning of clock cycle II.

This test verifies that the FFs or latches correctly latch the state driven by the first vector in the triplet, and that the state is not contaminated by the PPI stimulus itself. This procedure is illustrated in "Using PathMill to Detect PPIs" on page 64.

If you cannot verify this, it is suggested that you run GAP on the sequential design using the vector-pair approach (see tutorial 5).

Files Needed for this Tutorial

The following table lists the files needed for this tutorial. These files are located in the *pw_advanced/max_powr2* subdirectory in your working directory.

Filename	Description
cfg_max_power	Configuration file for maximum power estimation
max_seq_power.spi	SPICE netlist of the sequential circuit (a segment of a datapath with two pipelined stages) to be estimated
max_seq_power.vec	Vector file to be used for maximum power estimation
run_max_seq_power	Run script for maximum power estimation of sequential circuits
tech.typ.25c_5v	Technology file
runpm	Run script for latch detection by PathMill
cfgpm	Configuration file for latch detection using PathMill

Procedure

Use the following procedure to run the tutorial on estimating maximum power for sequential circuits.

1 Open and study the *cfg_max_power* configuration file.

The following file sample shows the contents of this configuration file.

```
use_vec_gap vec=max_seq_power.vec
set_gap_opt period=40ns save_history=on vec_history=1
guide_gap_search fit=2 level=7
set_ckt_cmd d-ff mark_node_latch no=q
print_probe_i inst gap.total_power
;report_block_powr stage1 track_power=1 x1*
;report_block_powr stage2 track_power=1 x2*
;report_block_powr ff_array track_power=1 xd*
;report_probe_i avg stage1.total_power
;report_probe_i avg stage2.total_power
;report_probe_i avg ff_array.total_power
;report_probe_i avg gap.total_power
```

Each line in this file is critical to the function of the GAP feature:

- `use_vec_gap vec=max_seq_power.vec`

  This line specifies the vector files for the GAP simulation. The GAP function generates a history file, as the simulation progresses, for each vector file. You can specify multiple vector files to be used with the GAP feature.

- `set_gap_opt period=40ns save_history=on vec_history=1`

  This command sets the following parameters for the GAP algorithm:

  - The `period=40ns` argument sets the clock period for the circuit to 40 ns.
  - The `save_history=on` argument saves the simulation history, which allows you to stop the simulation and

restart it from where you left off. This feature can come in handy when simulating large circuits, which can be very CPU intensive and often require a long simulation time. This option saves both the circuit state and vector stimuli, which allows you to do further power analysis on a specific generation without having to run the simulation again.

- ◆ The `vec_history=1` option saves the vector history for only those generations that improve (increase) the maximum power estimation.

■ `guide_gap_search fit=2 level=7`

This command sets up the fitness function and the search level. The `fit=2` option tells the simulator to maximize the instantaneous power (fit=1 maximizes average power per clock cycle).

NOTE: To ensure the quality of the resulting estimate, this tutorial uses a larger search level (7) than that used in tutorial 5 because the size of the solution space to be searched is larger than it is in tutorial 5.

■ `set_ckt_cmd d-ff mark_node_latch no=q`

This command enables the automatic searching and identification of latches and flip-flops in the circuit. The `d-ff` subcircuit is identified and the `mark_node_latch no=q` command is applied to all instances of the `d-ff` subcircuit. If you have a circuit with FFs that have many different subcircuit names, you have to apply a separate **set_ckt_cmd** command to each subcircuit. Specifying the **mark_node_latch** command causes the PPI vector file (*maxpower.ppi.vec*) to be generated and added to the simulation automatically.

■
```
print_probe_i inst gap.total_power
;report_block_powr stage1 track_power=1 x1*
;report_block_powr stage2 track_power=1 x2*
;report_block_powr ff_array track_power=1 xd
```

```
;report_probe_i avg stage1.total_power
;report_probe_i avg stage2.total_power
;report_probe_i avg ff_array.total_power
;report_probe_i avg gap.total_power
```

This section of the configuration file controls power reporting. All but the first line are commented out. See "Analyzing Generations with Saved State Files" on page 63 for information on using these power reporting commands.

NOTE: You can reduce the size of the resulting output file by using the FSDB or ISDB output format. To do so, simply add the `set_print_format for=isdb` (or `for=fsdb`) command to the configuration file. This setting does make the simulation run more slowly.

2 Open and study the *max_seq_power.spi* netlist file.

The described circuit contains four PPIs, which are the output of the four DFFs. Notice that these DFFs are 1) positive edge-triggered, 2) buffered at the output, and 3) instances defined by the subcircuit definition `d-ff`. The clock signal has a delay of 0 ns, a period of 40 ns, and a 50% duty cycle as described by the PULSE independent source function.

To work properly using the vector-triplet method, GAP needs to be synchronized with the clock signal driving the circuit. To achieve synchronization, you need to perform the following procedure:

- Apply the period= option with the **set_gap_opt** command to specify the period for the GAP estimation, which should be equal to the clock period (40 ns, in this tutorial), and
- Apply the delay= option with the **set_gap_opt** command to specify the delay for the GAP time window, which should be equal to the delay of the clock (0 ns, in this tutorial).

NOTE: You don't have to use the delay= option with the **set_gap_opt** command for this circuit, since the default starting time for GAP is 0.

The FF's clock-to-Q time is not considered in this tutorial (delay through the combinational part of the circuit is used to satisfy the hold time). If you have a circuit for which the clock-to-Q time needs to be taken into consideration, you would need to specify the values correctly before running GAP. For example, suppose the clock signal starts from 100 ns, and the clock-to-Q time of the FFs is 0.5 ns. To synchronize GAP's time window with the clock, you need to specify the delay= option with a value of 100 ns with the **set_gap_opt** command. In addition, to address the clock-to-Q time, you would have to set the delay= option of the **set_vec_opt** command to 0.5 ns.

3. Open and study the *max_seq_power.vec* vector file.

   Notice that this file contains only the PI signals; therefore, the **mark_node_latch** command is used to generate a vector file containing PPIs and add it to the GAP estimation automatically.

4. Run the simulation using the *run_max_seq_power* script.

   The autodetection simulation stops and restarts with the following message:

```
Reading configuration files...
PPI dumping has been completed. Adding vec file
maxpower.ppi.vec to GAP and restarting GAP
estimation.
```

5. When you see that the simulation has progressed somewhere beyond generation 4, type `Control-c` to stop the simulator in interactive mode.

6. Type `quit` to quit the simulation.

7. List the directory contents.

Notice that the GAP feature automatically saved a file with a name for similar to *maxpower.save.7200.000000ns.Z* for generation 3, which improved the maximum power.

You can use this compressed file to restart the GAP run from where you stopped it.

8 To restart (restore) the simulation from where you stopped it, add the following command to the configuration file:

```
restore_ckt_state 7200ns
```

9 Start the simulation again using the *run_max_seq_power* run script.

10 When the estimate is done, list the contents of the current directory.

The GAP estimation produces several output files with a *maxpower* prefix.

11 Open and examine the *maxpower.gap* file, which contains a summary of the GAP simulation (see Figure 17).

```
Estimates of maximum instantaneous power from the GAP
feature:
---------------------------------------------------
(only those generations increasing maximum power are
listed)

 Generation 0
 Maximum power: 0.080511 (watts)
 Vec[1]: 010000101101110011
 Vec[2]: 10001101011000zzzz
 Vec[3]: 11101011110111zzzz


 Generation 3
 Maximum power: 0.082563 (watts)
 Vec[1]: 101000001100110000
 Vec[2]: 00100010010010zzzz
 Vec[3]: 10011001101001zzzz

..........................

 Generation 92
 Maximum power: 0.120024 (watts)
 Vec[1]: 000110100110111011
 Vec[2]: 00000000010000zzzz
 Vec[3]: 11111111101111zzzz


 Generation 93
 Maximum power: 0.121689 (watts)
 Vec[1]: 000110100010110001
 Vec[2]: 00000000000000zzzz
 Vec[3]: 11111111111111zzzz

------------------------------------------------------
Total number of generations: 256
Total CPU time             : 1244.61 (secs)
```

Figure 17 Contents (partial) of the maxpower.gap file

Each generation of the GAP estimation simulates the circuit with 20 input vector triplets (by default). The *maxpower.gap* file contains an entry for each generation that improved the maximum power. Each entry consists of the following fields: generation ID, maximum power produced, and the input vector triplet producing the maximum power. The total CPU time and total number of generations are printed at the end of the file.

12 Open and examine the *max_seq_power.gav* and *maxpower.ppi.gav* files.

These are the history files for the *max_seq_power.vec* and *maxpower.ppi.vec* vector files, respectively. The two *.gav* files contain the vectors for all generations that increased the maximum power. You can use the vec_history= option of the **set_gap_opt** command to specify which vectors should be kept.

See the *PowerMill Reference Guide* for details on all of the GAP configuration and interactive commands.

NOTE: You can use the *.gav* files (generated by GAP) as vector files for a regular PowerMill simulation (by specifying the -nvec command-line option). However, if you want to fully reproduce the GAP result for a specific generation, you have to consider the state of the circuit; therefore, you need to use the save and restore mechanism (previously described in this procedure).

13 Load the *maxpower.out* file in turboWave and view the mathematical signal m(gap.total_power), which represents the instantaneous power (in watts) of the sequential circuit used in this tutorial (see Figure 18).

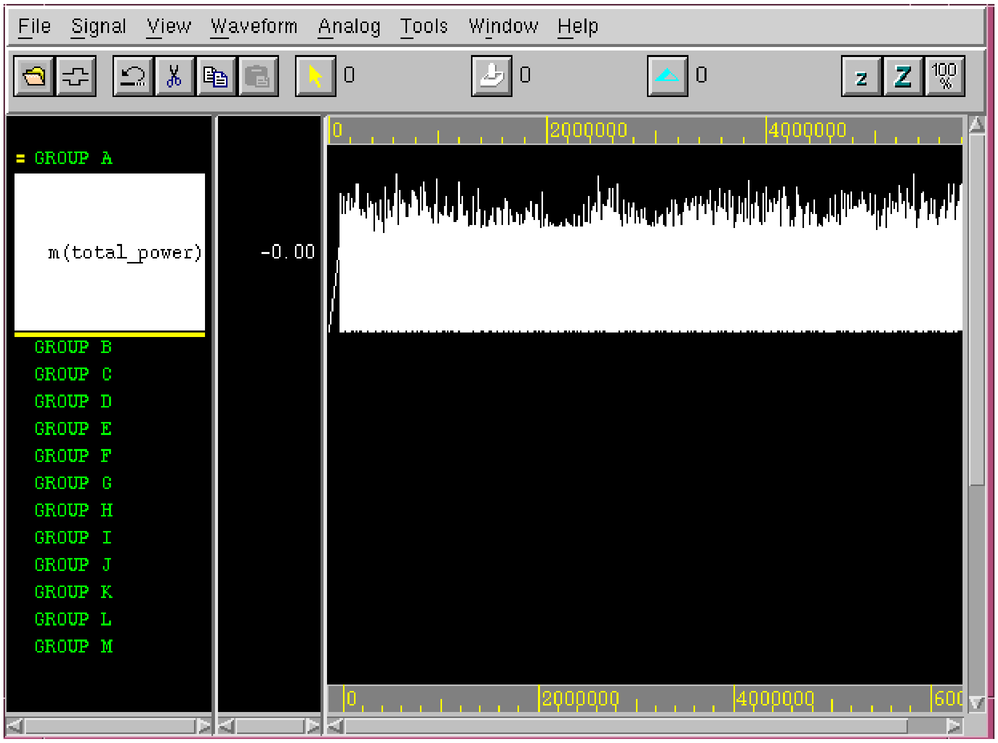


Figure 18 Waveform of m(gap.total_power) signal in maxpower.out file

14 After a GAP estimation, you can use the generated state files, which have a *.Z* extension, in two ways:

- You can resume a GAP estimation from the time the state file was saved.
- You can perform further analysis on those generations with saved state files without re-running the whole simulation (see "Analyzing Generations with Saved State Files" on page 63).

Analyzing Generations with Saved State Files

The following procedure shows you how to analyze the average power for different blocks within the two-stage pipeline circuit for generation 3 (with a state file named *maxpower.save.7200.000000ns.Z*).

1. Verify that the following command has been added to the configuration file:

   ```
   restore_ckt_state 7200ns
   ```

2. In the configuration file, uncomment the last seven commands, which are currently commented out.

   These commands create three blocks within the circuit (the 4-bit adder at the first stage, the 4-bit adder at the second stage, and the FFs), and report the average power for individual blocks.

3. Run the simulation using the *run_max_seq_power* script.

   When you see that the simulation has progressed somewhere beyond generation 3, type `Control-c` to stop the simulator in the interactive mode.

4. Type `guide_gap_search max_gen=1` at the interactive prompt.

5. Resume the simulation by typing `cont_sim`.

   The simulation stops right after generation 3 is finished.

6. Type `quit` to end the simulation and exit.

7 Open and examine the *maxpower.log* file.

```
Current information calculated over the intervals:

     0.00000e+00 -  2.40010e+03 ns

Node: stage1.total_power
    Average power       :  7.03574e+02 uW

Node: stage2.total_power
    Average power       :  7.87411e+02 uW

Node: ff_array.total_power
    Average power       :  9.66177e+03 uW

Node: gap.total_power
    Average power       :  1.11595e+04 uW
```

This file contains power reports for the following four blocks:

- The 4-bit adder at the first stage (`stage1.total_power`)
- The 4-bit adder at the second stage (`stage2.total_power`)
- The flip-flops (`ff_array.total_power`)
- The whole circuit (`gap.total_power`)

This report also includes various current statistics (for example, average and RMS supply current, and average and RMS input current) for individual blocks.

Using PathMill to Detect PPIs

In this tutorial, the circuit description (in the *max_seq_power.ppi* file) contains a subcircuit definition for FFs, therefore you can use the **mark_node_latch** and **set_ckt_cmd** commands to do PPI detection automatically. However, if your circuit does not contain a subcircuit (or pattern) definition for FFs or latches and you have a copy of the PathMill simulator, you can use PathMill's latch detection feature to generate a PPI vector file that you can use with a GAP estimation.

NOTE: If you use PathMill for PPI detection, you must verify the functionality of the latches by running an additional GAP simulation (see “Testing Latch Functionality for Flip-Flops with Non-Buffered Outputs” on page 53).

Use the following procedure to generate a *.ppi.vec* file using the PathMill simulator:

1 Open and study the *cfgpm* configuration file.

The file contains only one PathMill configuration command—**pw_detect_ckt_latch**. This command instructs PathMill to perform latch detection on the circuit and generate a vector file containing the output of all the detected latches.

See the command definition for **pw_detect_ckt_latch** in the *PathMill Reference Guide*.

2 Run PathMill using the *runpm* script, which consists of the following PathMill command line:

```
pathmill -n max_seq_power.spi -c cfgpm
```

The latch-detection simulation starts and stops with the following messages:

```
Start Latch Detection
Latch Detection completed
```

3 Open and study the *pathmill.ppi.vec* file. The following file sample shows its contents.

```
signal XD1.c XD1.f XD2.c XD2.f XD3.c XD3.f XD4.c XD4.f
type VEC
io pppppppp
```

The circuit described in *max_seq_power.spi* contains four DFFs. Since each flip-flop in this circuit is comprised of two cascaded latches, the *pathmill.ppi.vec* file includes eight signals. Notice that the io type for all eight signals is p (pseudo-primary input).

NOTE: The signals in any GAP vector file can have one of three different input/output types: i (input), b (biput), and p (pseudo-primary input). Open the *cfg_max_power* file.

4 Comment out the following line:

```
set_ckt_cmd d-ff mark_node_latch no=q
```

5 Add the *pathmill.ppi.vec* file, generated by PathMill, to the PowerMill GAP simulation using the vec= option of the **use_vec_gap** command:

```
use_vec_gap vec=max_seq_power.vec
vec=pathmill.ppi.vec
```

6 Add the following command to the *cfg_max_power* file to verify the functionality of the FFs or latches.

```
print_node_logic s1[*] d[*]
```

7 Add the `max_gen=1` option to the end of the `guide_gap_search` command specification:

```
guide_gap_search fit=2 level=7 max_gen=1
```

Setting the `max_gen=` option to 1 forces the GAP simulation to stop after one generation.

This circuit contains only one type of FF, and the s1[*] and d[*] nodes are the inputs and outputs for the FFs.

8 Run the simulation using the *run_max_seq_power* script.

9 After the simulation stops, type `quit` to get out of interactive mode and the simulation.

10 Load the *maxpower.out* file in turboWave and view the two bus signals d[3:0] and s1[3:0].

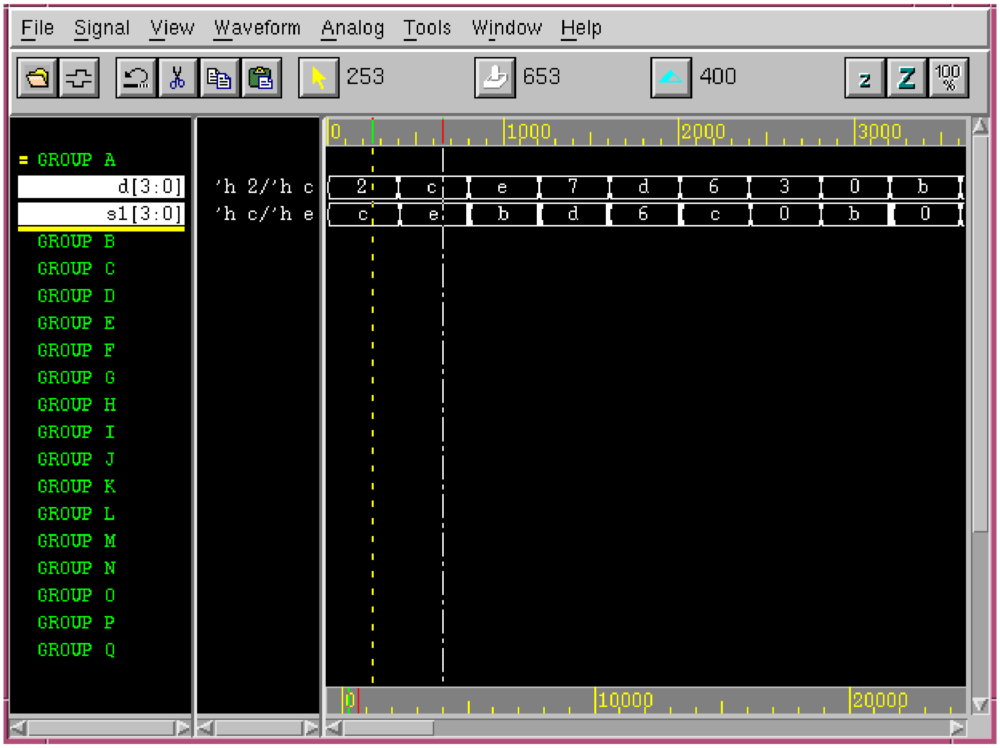


Figure 19 Waveform of bus signals d[3:0] and s1[3:0] in the maxpower.out file

For the first vector triplet, the value of s1[3:0] at the end of clock cycle I (“c”) is equal to the value of d[3:0] at the beginning of clock cycle II. This is also true for the second and third vector triplets, which verifies the functionality of the FFs or latches as described in the section, “Testing Latch Functionality for Flip-Flops with Non-Buffered Outputs” on page 53.

11 Open the *cfg_max_power* file.

12 Remove the following line:

```
print_node_logic s1[*] d[*]
```

13 Remove the `max_gen=1` argument from the `guide_gap_search` command specification.

14 Run the regular GAP simulation using the *run_max_seq_power* script.

At this point, you can resume the regular GAP procedure starting with step 5 on page 58.

NOTE: When you use PathMill, instead of PowerMill, to detect the PPIs the *maxpower.ppi.vec* file is replaced by the *pathmill.ppi.vec* file; therefore, the corresponding *maxpower.ppi.gav* is replaced by the *pathmill.ppi.gav* file.

Tutorial 7: Customizing a GAP Objective Function using an ADFMI Code File

This tutorial illustrates an advanced GAP feature which allows you to customize the objective function using an ADFMI C file. Using this feature, you can create your own objective function to be maximized by GAP, instead of using the two default objective functions.

A customized objective function is a regular C function in an ADFMI C file. You can use various circuit parameters (for example, node voltage, node current, element branch current, and block power) to calculate the fitness measure.

The following table lists the files needed for this tutorial. These files are located in the *pw_advanced/max_powr3* subdirectory in your working directory.

Filename	Description
cfg_max_power	Configuration file for maximum power estimation
max_seq_power.spi	SPICE netlist of the sequential circuit (a segment of a datapath with two pipelined stages) to be estimated
max_seq_power.vec	Vector file to be used for maximum power estimation
cmos35t.mod	A BSIM3 v3 SPICE model used to run the simulation in direct mode
gap_customize.c	An ADFMI C file used to customize the objective function for the GAP simulation
run_custom	Run script for GAP customization

Procedure

Use the following procedure to customize the objective function for a GAP simulation.

1 Open and study the *max_seq_power.spi* file.

Notice that the clock is defined at the bottom of the file as a pulse signal with a delay of 100ns and a period of 40ns. The circuit contains four FFs (positive edge-triggered) which are instances defined by the subcircuit `d-ff`. The output of the subcircuit `d-ff` is buffered.

2 Open and study the *cfg_max_powr* configuration file. The following file sample shows the contents of this file.

```
use_vec_gap vec=max_seq_power.vec
set_gap_opt delay=100ns period=40ns vec_history=2
guide_gap_search fit=3 level=7
set_ckt_cmd d-ff mark_node_latch no=q
print_probe_i inst customize_gap_fitness
```

Each line in this file is critical to the function of the GAP feature:

- `use_vec_gap vec=max_seq_power.vec`

  As in tutorial 6, the vector file max_seq_power.vec contains only the PIs. This command registers this vector file to GAP estimation. A vector history is tracked for this vector file as the simulation progresses.

- `set_gap_opt delay=100ns period=40ns vec_history=2`

  This command synchronizes GAP's time window with the clock driving the circuit by specifying a delay of `100ns` and a period of `40ns`. This command also uses the vec_history= option to tell the simulator to track the vector history for all generations that improve the maximum value of the fitness function. Notice that the save_history= option is disabled; therefore, no state files, printed with a .Z extension, are generated for this GAP simulation.

NOTE: The clock-to-Q time is not considered for the circuit in this tutorial since delay through the combinational part of it can satisfy the hold time of the FFs. Otherwise, you would need to set the delay= option, of the **set_vec_opt** command to be equal to the clock-to-Q time.

- `guide_gap_search fit=3 level=7`

  This command enables GAP customization by setting the fit= option to 3, which notifies GAP that the average value of the fitness function (defined in an ADFMI C file) should be the objective to maximize. This command also sets the search level option (level=) to 7.

- `set_ckt_cmd d-ff mark_node_latch no=q`

  This line enables GAP's built-in latch detection mechanism to generate and register the PPI vector file with the GAP feature. The `mark_node_latch` command searches nodes that have the name `q` under all instances of subcircuit `d-ff`. Notice that the name of the output of subcircuit `d-ff` is `q`. A PPI vector file containing these nodes is automatically generated and registered with the GAP simulation.

- `print_probe_i inst customize_gap_fitness`

  This command instructs GAP to generate the waveform of the value of the customized objective function (vs. time), and print it to the output file. The name `customize_gap_fitness` is the default name for the math signal representing the value of the customized objective function. You can also choose to print out the average and RMS values of this signal (see the definition of the **print_probe_i** command in the *PowerMill Reference Guide*).

3 Open the *gap_customize.c* file.

This is the ADFMI C file in which you define and register your customized fitness function. See "Creating a Customized Fitness Function" on page 108 in Chapter 4 for instructions on creating this file).

4 Locate the lines that are part of the **RegisterUserModel()** function at the bottom of the file (see Figure 20).

```
/* Specify block power for block1 and block2 */
/* block1 will be the first bank of 4 adders */
/* and block2 the second bank of 4 adders */
fmConfigCmd ("report_block_powr block1 track_power=1 x1*");
fmConfigCmd ("report_block_powr block2 track_power=1 x2*");
/* Register arguments and function pointer of customized
objective function */
fmRegisterGap("p(block1) p(block2)", fitness);
```

Figure 20 Excerpt from registration function

The first two lines use the `fmConfigCmd()` function to specify a PowerMill configuration command that maps the `x1*` and `x2*` blocks to the `block1` and `block2` power-tracking variables. They also tell the simulator what to track by setting the track_power= argument to 1.

Next, the `fmRegisterGap()` function maps the power of block1 and block2 variables into the GAP algorithm. They are used in the fitness function.

NOTE: The asterisk (*) wild card specifies that all four adders in each bank are to be included in the block specifications.

Find the execution function at the top of the file (see Figure 21).

```
double fitness()
{
  static int id_4_block1_powr, id_4_block2_powr;
  double sum = 0;

  if (fmSimPhase() == FMDCINITPHASE) {
    /* get id for signals, which can be used later to
       access their value */
    id_4_block1_powr = fmGapGetSignalId ("p(block1)");
    id_4_block2_powr = fmGapGetSignalId ("p(block2)");
    return 0;
  }
  else {
    /* calculate fitness measure */
    sum =
fmGapGetValueById(id_4_block1_powr)+fmGapGetValueById(id_4
_block2_powr);
    return sum;
  }
}
```

Figure 21 Excerpt from execution function

The first section of this function checks to see if the simulator is in the DC initialization phase prior to the start of the transient simulation. If so, it sets the ID pointers. If it is not in the DC initialization phase, the transient simulation has already started.

The second section calculates the sum variable by adding up the power from both blocks.

Notice that the power number you get is much lower than in the previous tutorial, in which you calculated the maximum power for the entire circuit. In contrast, this tutorial only covers two blocks of the circuit (see Figure 22).

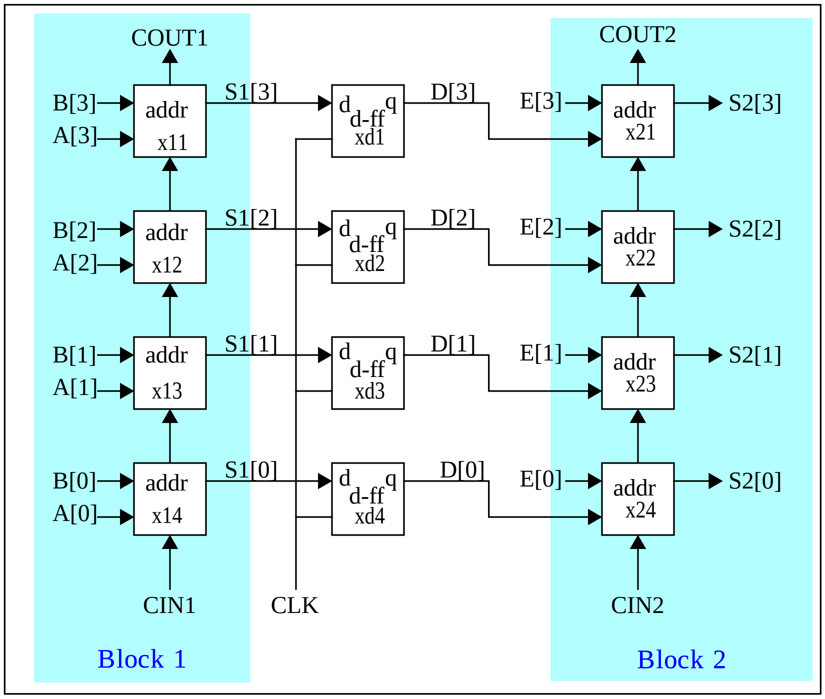


Figure 22 Block diagram of the circuit

5 Open and study the *run_custom* script.

6 Locate the following option:

```
-fm gap_customize.c
```

This option identifies the *gap_customize.c* file as an ADFMI C file. In addition to describing and registering ADFMI models, an ADFMI C file can also describe and register a GAP customized fitness function. Specifying the ADFMI C file on the command line is required for GAP customization.

7 Run the simulation using the *run_custom* script.

The simulation performs PPI detection first. After it generates the *custom.ppi.vec* PPI vector file, it registers it with GAP simulation.

The following messages appears on the screen:

```
Reading configuration files ...
```

```
PPI dumping has been completed. Adding vec file
custom.ppi.vec to GAP and restarting GAP estimation.
```

The simulation resumes automatically in GAP mode.

8 When you see that the simulation has progressed somewhere beyond generation 8, type `Control-c` to stop the simulator in interactive mode.

9 Type `report_gap_parameters` at the interactive prompt.

This interactive command prints a status report of GAP parameters to the screen (see Figure 23).

```
Current settings of parameters/options for GAP:

 Fitness function is peak average power
 History option                                : 2
 Population size                               : 20
 Search_level                                  : 7
 Clock period                                  : 40 (ns)
 No upper limit on CPU time
 No upper limit on number of generations
 No upper limit on number of vector triplets
 No upper limit on value of fitness function
```

Figure 23 Report of GAP parameter settings

This command allows you to view and check the current setting of the GAP parameters, including the vector history option, population size, search level, and various stopping criteria (for example, CPU time, number of generations, and number of triplets).

10 Type `report_gap_progress` at the interactive prompt. This command prints a progress report for the GAP simulation to the screen (see Figure 24).

```
GAP result so far:

time (secs)     num. of triplets    max power (watts)
--------------------------------------------------
 6.92                 20          0.000714735

 Vec[1] : 010000101101110011
 Vec[2] : 10001101011000zzzz
 Vec[3] : 11101011110111zzzz

 10.49                 40          0.000759248

 Vec[1] : 101101111000110111
 Vec[2] : 01000101100110zzzz
 Vec[3] : 10010010111111zzzz


 ......................................


  21.29                100          0.000938212

 Vec[1] : 101101110001110101
 Vec[2] : 01000101011000zzzz
 Vec[3] : 10010011110111zzzz

 24.96                120          0.00111505

 Vec[1] : 001101110111000010
 Vec[2] : 01000101010011zzzz
 Vec[3] : 10010010111101zzzz

--------------------------------------------------
Current CPU Time               : 33.47 (secs)
Maximum power so far           : 0.00111505 (watts)
Current number of generations  : 8
```

Figure 24 Report of GAP parameter settings

The simulator generates an entry in this file for each generation that improves the maximum value of the fitness function. Each file entry lists the time, number of triplets, current value of the fitness function, and the stimuli.

In addition, the current CPU time, maximum value, and number of generations are printed at the bottom of the report.

11 You can also use the **guide_gap_search** command to set various criteria for stopping GAP at the interactive mode command line. See the command definition for the **guide_gap_search** command in the *PowerMill Reference Guide.*

12 Type `cont` to continue the GAP simulation.

13 When the simulation finishes, type `quit` to quit the simulation.

14 Open and study the *custom.gap* file (see Figure 25).

The simulator creates an entry in this summary file for each generation that improves the maximum value.

15 Open and study the *max_seq_power.gav* and *custom.ppi.gav* files. These two files are the vector history files for *max_seq_power.vec* and *custom.ppi.vec*, respectively.

The *custom.ppi.vec* file is the PPI vector file generated by the **mark_node_latch** command. Since the vec_history= option of the **set_gap_opt** command is specified as 2, stimuli of all the generations that improved the maximum value are tracked in the two *.gav* file.

```
Estimates of maximum average power from the GAP feature:
-------------------------------------------------------
----
(only those generations increasing maximum power are
listed)

 Generation 0
 Maximum power: 0.000714735 (watts)
 Vec[1]: 010000101101110011
 Vec[2]: 10001101011000zzzz
 Vec[3]: 11101011110111zzzz


 Generation 1
 Maximum power: 0.000759248 (watts)
 Vec[1]: 101101111000110111
 Vec[2]: 01000101100110zzzz
 Vec[3]: 10010010111111zzzz


 Generation 2
 Maximum power: 0.000882713 (watts)
 Vec[1]: 101101111000110111
 Vec[2]: 01000101000110zzzz
 Vec[3]: 10010010111111zzzz


 Generation 3
 Maximum power: 0.000904537 (watts)
 Vec[1]: 101101111111000110
 Vec[2]: 01000101010011zzzz
 Vec[3]: 10010010111101zzzz
```

Figure 25 Contents (partial) of the custom.gap file

Tutorial 8: Finding Static Leakage Paths

This tutorial illustrates how you can use PowerMill to find static DC short circuit paths. A static DC short circuit path exists if a node has a static conducting path to VDD and a static conducting path to GND. The **report_ckt_leak** command finds static DC paths and detects several unusual topological conditions in regular CMOS designs. Various topologies are listed in the *PowerMill Reference Guide* under the description of the **report_ckt_leak** command. This tutorial highlights the effect of each of these topologies.

The following table lists the files needed for this tutorial. These files are located in the *pw_advanced/static_leak* subdirectory in your working directory.

Filename	Description
cfg_static	Configuration files for static leak analysis
static_powr.spi	SPICE netlist of the circuit for block power analysis (see Figure 26)
run_static	Run scripts for static leak tutorial
tech.typ.25c_5v	Technology file

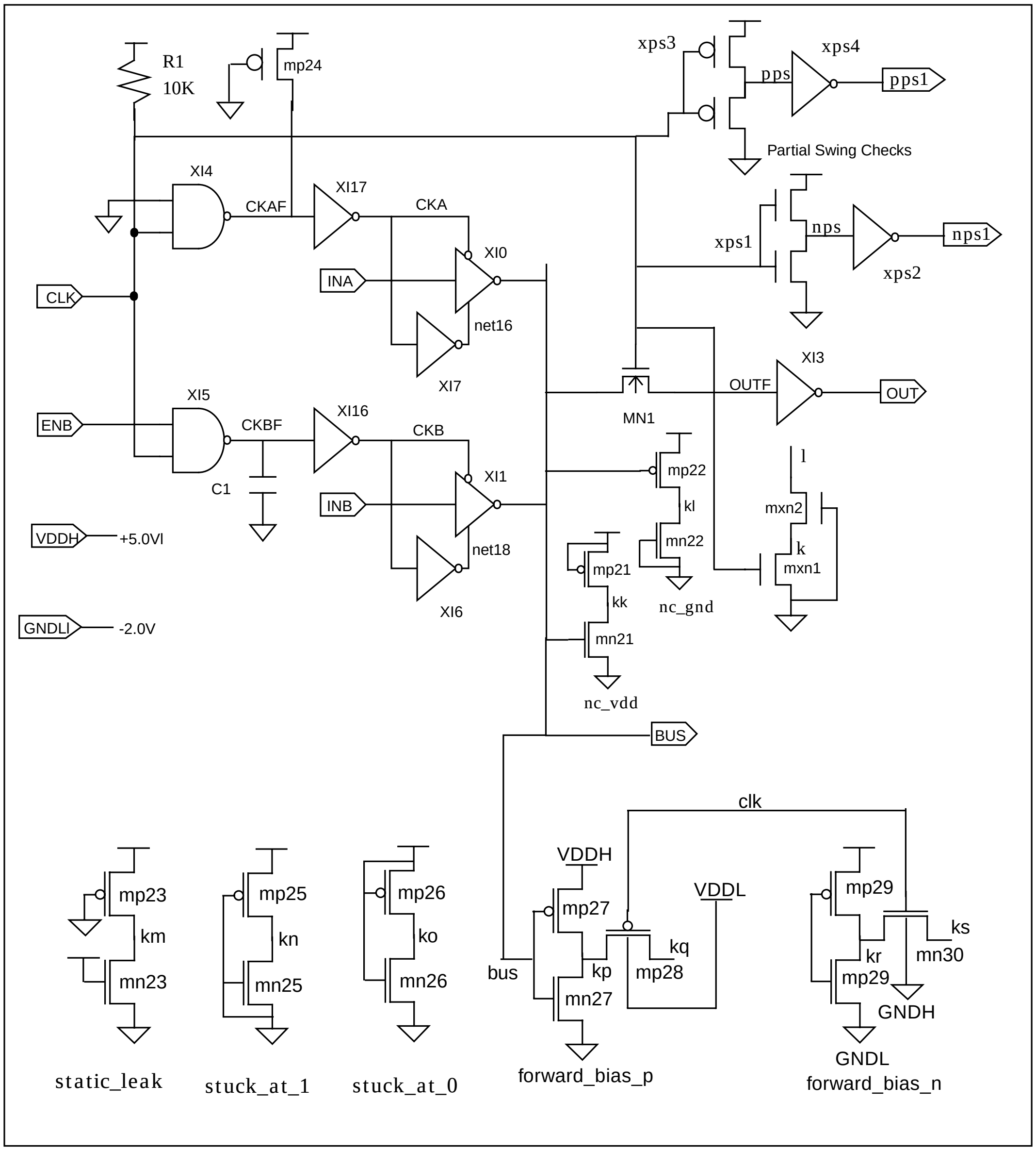


Figure 26 Schematic of the static_powr.spi netlist

Procedure

Use the following procedure to run the static leakage path check tutorial.

1 Open and examine the *cfg_static* configuration file (see Figure 27).

```
set_sim_case l
print_node_v *
;report_ckt_leak el=* to_vdd to_gnd
;report_ckt_leak el=* static_leak
;report_ckt_leak el=* stuck_at_1 stuck_at_0
;report_ckt_leak el=* partial_off_n partial_off_p
;report_ckt_leak el=* n_path p_path
;report_ckt_leak el=* nc_vdd nc_gnd
;report_ckt_leak el=* forward_bias_n forward_bias_p
report_ckt_leak el=*
set_node_thresh 0.5 2.5 v=1 gnd* vdd*
```

Figure 27 Contents of the cfg_static file

2 Run the simulation using the *run_static* script.

3 When the simulation is finished, list the directory contents.

Notice that the directory now contains several output files with the static prefix.

4 Open the static.log file and notice the number of errors reported in the *.err* file.

5 Use the viewerror utility to view the errors reported in the *static.err* file.

```
viewerror -i static.err -o all_error
```

The *all_error* file should list 25 errors. This file lists the type of error followed by a brief description of why the error occurred. It also lists the transistors that are present in the static leakage path.

6 Study each type of error and go back to the netlist and figure out why they are occurring. Also try to match the topologies present in the netlist with those detailed in the reference guide.

NOTE: The **set_node_thresh** command is important and should be used while doing static leakage analysis. This is because the static analysis uses the node voltages and logic thresholds for the constant voltage source nodes to identify which nodes are ground and which nodes are supply (VDD).

Although, the voltage in the technology file is 5.0 V, and the VDD is 3.0 V, the simulator sets the thresholds to more reasonable values by checking the neighboring connection. However, it is good practice to always use the **set_node_thresh** command to set appropriate thresholds.

7. Comment out all the last **report_ckt_leak** command (`report_ckt_leak *`).

8. Uncomment each of the other **report_ckt_leak** commands one by one and rerun the simulation each time.

   This process allows you to see the error messages that are particular to different types of topologies.

NOTE: The **report_ckt_leak** command is set up so that if one possible error presumes others, only the most serious error is reported. For example, if a node is stuck_at_1, it is equivalent to saying that it has a static path to VDD and no path to gnd. If the stuck_at_1 error is reported, the nc_gnd and to_vdd errors are not reported since they are redundant.

Chapter 4

Power Analysis

Overview

This chapter addresses the various features provided by PowerMill for the analysis and diagnosis of power use for your circuits and includes information on the following subjects:

- Analyzing Power Consumption
- Using the GAP Feature to Estimate Maximum Power
- Performing a Dynamic Power Consumption Analysis
- Performing a Static Power Consumption Diagnosis
- Analyzing Power Using the RC and UD Simulation Modes

In discussing PowerMill simulation results, the term *power* is synonymous with *current*.

Analyzing Power Consumption

This section provides an overview of the features and methods available in PowerMill for the analysis of power use in your circuits at the full-chip, block, and individual node or element levels.

This section includes information on the following subjects:

- Analyzing Power at Full-Chip Level
- Analyzing Power Block-by-Block
- Measuring Wasted Power
- Measuring True Power in Watts
- Measuring Power Hierarchically
- Assigning Power Budgets
- Retrieving Power Information Interactively
- Controlling Power Reporting Resolution and Accuracy
- Printing and Reporting Branch Currents

- Printing and Reporting Internal Node Currents
- Working with Probes
- Current Histograms and Tracking Windows

Analyzing Power at Full-Chip Level

Full-chip power analysis is the simplest type of power analysis. At this level the power usage of the chip can be characterized by the current statistics and waveforms of the main power nodes, such as VDD and GND. These can be obtained by using the node current printing and reporting configuration commands.

print_node_i	**print_node_didt**
print_node_iavg	**report_node_i**
print_node_irms	**report_node_powr**

When used for any voltage source node the current reported is the current being supplied by the source (see "Printing and Reporting Internal Node Currents" on page 99).

The **print_node_i*** commands are used to print the instantaneous, average, RMS, and didt waveforms of the requested nodes to the *.out* file. The **report_node_i** command can be used to request the reporting of the average, RMS, or peak currents to the *.log* file at the conclusion of the simulation. You can also use it to request a current histogram to be generated for the specified nodes (see "Current Histograms and Tracking Windows" on page 101). The **report_node_powr** command is a macro and can be used to request all of the above information (except the didt waveform and current histogram) with one command.

The simplest approach is to use the **report_node_powr** command without specifying any arguments. This automatically selects all the supply and ground nodes, as well as all current probes, for current reporting.

Figure 1 shows a sample report generated by the **report_node_powr** command. This report is first printed to the screen and then printed in the *.log* file, at the end of the simulation. The instantaneous, average, and RMS current waveforms for these nodes are also recorded in the *.out* file.

```
Current information calculated over the intervals:

    0.00000e+00 -  6.00000e+03 ns

Node: gnd
    Average current      :  7.11217e+02 uA
    RMS current          :  1.26582e+03 uA

    Current peak #1      :  1.49260e+04 uA  at   4.40180e+03 ns
    Current peak #2      :  1.40570e+04 uA  at   4.20180e+03 ns
    Current peak #3      :  1.32660e+04 uA  at   4.00180e+03 ns
    Current peak #4      :  1.32540e+04 uA  at   3.80180e+03 ns
    Current peak #5      :  1.31900e+04 uA  at   2.86860e+03 ns

Node: vdd
    Average current      : -7.09839e+02 uA
    RMS current          :  1.26431e+03 uA

    Current peak #1      : -1.54890e+04 uA  at   4.40181e+03 ns
    Current peak #2      : -1.46190e+04 uA  at   4.20181e+03 ns
    Current peak #3      : -1.38260e+04 uA  at   4.00181e+03 ns
    Current peak #4      : -1.38160e+04 uA  at   3.80181e+03 ns
    Current peak #5      : -1.32800e+04 uA  at   2.86860e+03 ns
```

Figure 1 Sample power report (generated by report_node_powr)

If you prefer to monitor power consumption during the simulation, you can use either the turboWave and SimWave waveform viewer to watch the power consumption during the simulation.

For more specific information on each command, see Chapter 2, "Configuration and Interactive Commands" in the *PowerMill Reference Guide.*

Analyzing Power Block-by-Block

Block-level power analysis allows the you to find out where most of the power is being consumed within a circuit. With this information, you can optimize power within blocks that consume too much power. Using a hierarchical netlist (a flat netlist will not work) you can isolate power to portions of blocks, if necessary. This type of analysis should be done in the design phase, prior to finishing the layout.

A block-level power analysis is most easily done using the **report_block_powr** configuration command.

EXAMPLE:

```
report_block_powr my_block x1.x1.*
```

This example generates a block power report to the *.log* file for the subcircuit `x1.x1` under the label `my_block`.

The block power report provides a statistical analysis of the block including the number of elements, internal nodes, source nodes, and biput nodes. By default, it also lists the average and RMS source, ground, input, output, and biput currents for the block.

See “Sample Power Reports” on page 223 in Appendix A for a sample block power report. This report is first printed to the screen and then reported in the *.log* file, at the end of the simulation. Using optional arguments, the **report_block_powr** command can report the average and RMS ground current, and produce a list of the current supplied by each individual supply, ground, input, output, and biput node.

You can also produce a block-level waveform and current histogram. For more details, see the *PowerMill Reference Guide*, and “Working with Probes” on page 100.

Measuring Wasted Power

PowerMill defines wasted power, also called short-circuit current, for a circuit block, as any DC current consumed by the circuit block that is not used to charge block capacitances or capacitances driven by the block. Specifically, PowerMill defines the total current consumed by the block as the sum of the total supply current and the current from the biputs that are acting as supplies. (The current from biputs acting as ground is not included.)

```
consumed current = total supply + Σ biput currents (i < 0)
```

The consumed current is comprised of two types of current: capacitive current and wasted current:

- **Capacitive Current:** any current for which its conductive path terminates (or originates) at a circuit capacitance. This current is essential in carrying out the switching functions of the circuit.
- **Wasted Current:** any current for which both ends of the conducting path are voltage sources. This current does not contribute to the changing of circuit voltages and, therefore, is considered “wasted.”

  Usually, wasted power consists of the DC current flowing from power to ground. This wasted power can come from crowbar or short-circuit current caused by both P-channel and N-channel devices being on the during a transition. It can originate from floating inputs or from DC paths that are unintended. It can also come from floating inputs or from DC path that are unintended such as device leakage currents. It can also come from analog bias circuits and current mirrors in operational amplifiers that cannot be avoided, but will be reported as wasted power.

Using the report_block_powr Command

You can measure wasted power using the track_wasted= option of the **report_block_powr** command.

EXAMPLE:

```
report_block_powr my_block track_wasted=1 x1.x1.*
```

This command generates a power analysis report for the subcircuit `x1.x1.*` that includes both the standard block power report (see “Analyzing Power Block-by-Block”) and a wasted power report.

Figure 2 shows a sample of the additional information provided by the track_wasted= option. This report provides the following information about the block:

- **Average and RMS capacitive current**

  The total current expended in charging block capacitances.

- **Average and RMS wasted current**

  The total DC current consumed by the block and not used to charge the block capacitances.

- **Leakage Percentage**

  The fraction of current consumed by the block, from all sources, that is not used to charge block capacitances.

```
Average capacitive current: -39057.934468 uA
RMS capacitive current    :  56567.265346 uA

Average wasted current     : -6115.921641 uA
RMS wasted current         :  9764.697468 uA

Wasted current percentage  :  13.538631%
```

Figure 2 Sample information provided by the track_wasted= option of the **report_block_powr** command

You can also produce a block-level waveforms and current histograms for the capacitive and wasted currents. For more details, see the *PowerMill Reference Guide*, and "Working with Probes" on page 100.

Measuring True Power in Watts

Using PowerMill, you can obtain the true power consumption (in watts) for a given circuit block. To do this, you need to specify the track_power= option when using the **report_block_powr** command.

EXAMPLE:

```
report_block_powr my_block track_power=1 x1.x1.*
```

In addition to the basic current reports, this command generates, a report of the total power consumption of the block. The power reported includes contributions from all sources and biputs for the block.

```
Average block power                    :  179937.712787 uW
RMS block power                        :  259344.959596 uW
```

Figure 3 Sample information provided by the track_power= option of the **report_block_powr** command

You can also produce block-level waveforms and current histograms. For more details, see the *PowerMill Reference Guide*, and “Working with Probes” on page 100.

Measuring Power Hierarchically

The level= and at_level options of the **report_block_powr** command allow you to do an analysis of block-level power in a hierarchical netlist. You can use these options to request power reports for all hierarchical sub-blocks of the specified circuit block with a single command.

The level= option automatically generates power reports for all sub-blocks of the specified subcircuit down to the specified level.

EXAMPLE 1:

```
report_block_powr top level=2 *
```

This command generates a power report for the entire circuit, and automatically generates reports for all sub-blocks down to level 2, such as x1.*, x2.*, x1.x1.* and, x1.x2.*.

The at_level= option generates power reports only for the sub-blocks at the specified level.

EXAMPLE 2:

```
report_block_powr top at_level=2 *
```

This command generates power reports for the entire circuit and the second-level sub-blocks x1.x1.* and x1.x2.* but not the first level sub-blocks.

NOTE: In each case, the specified level refers to the absolute level in the circuit hierarchy, and is not relative to the level of the specified pattern.

EXAMPLE 3:

```
report_block_powr my_block at_level=4 x1.x2.*
```

This command generates a power report for the `x1.x2.*` sub-block and its fourth-level hierarchal sub-blocks, such as x1.x2.x3.x4.*.

For each sub-block generated using the level= or at_level= options, PowerMill writes a standard block power report to the *.log* file. In each case, the reporting options will be the same as the top-level block. In addition, when either option is used, the simulator writes a hierarchical power report for the specified block hierarchy to the *.power* file. The file sample in Figure 4 shows a report for the total supply current as printed to the *.power* file.

PowerMill generates a similar report for the ground, supply, biput, wasted currents, and the power if those quantities are being tracked.

```
BLOCK top: AVERAGE SUPPLY CURRENT.

    LEVEL                CURRENT      PERCENT OF     PERCENT OF     CHILD BLOCK NAME
                          (uA)          PARENT          TOP
------------------------------------------------------------------------------

*---------     0        -1537.8       100.00         100.00        top
-*--------     1        -1537.8       100.00         100.00        xsr34x22.top
--*-------     2        -359.83        23.40          23.40        xsr34x22.x1161.top
--*-------     2        -322.71        20.98          20.98        xsr34x22.x1164.top
--*-------     2        -214.54        13.95          13.95        xsr34x22.x1166.top
--*-------     2        -207.33        13.48          13.48        xsr34x22.x1158.top
--*-------     2        -192.92        12.55          12.55        xsr34x22.x1162.top
--*-------     2        -81.777         5.32           5.32        xsr34x22.x1163.top
--*-------     2        -57.569         3.74           3.74        xsr34x22.x1157.top
--*-------     2        -52.966         3.44           3.44        xsr34x22.x1160.top
--*-------     2        -42.359         2.75           2.75        xsr34x22.x1169.top
--*-------     2        -5.8502         0.38           0.38        xsr34x22.x1167.top
--*-------     2       0.0086932       -0.00          -0.00        xsr34x22.x1170.top
--*-------     2               0       -0.00          -0.00        xsr34x22.x1165.top
--*-------     2               0       -0.00          -0.00        xsr34x22.x1168.top
--*-------     2               0       -0.00          -0.00        xsr34x22.x1159.top
```

Figure 4 Sample supply current report (from the .power file)

In addition, you can generate a block-level waveform and current histogram for each of the sub-block reports. For details, see the *PowerMill Reference Guide*, and "Working with Probes" on page 100.

Assigning Power Budgets

You can use PowerMill to assign power/current budgets to blocks created with the **report_block_powr** command and monitor those blocks for violations.

You will need to use the **set_block_budget** command to do this.

EXAMPLE 1:

```
report_block_powr top *
set_block_budget max_avg_src=100uA max_inst_src=1mA
   top
```

This pair of commands creates a block labeled top which consists of the entire circuit, and assigns current budgets of 100 uA for the average supply current, and 1 mA for the instantaneous supply current.

Violations of the assigned budgets are reported as errors to the *.err* file and reported to the *.power* file. See Appendix A for a sample *.power* file.

You can assign budgets to the instantaneous, average, and RMS values of the block supply current, ground current, biput current, wasted current, and power, as well as to the block's leakage percent.

Ordinarily, PowerMill only checks for budget violations at the conclusion of the dynamic simulation. However, if you specify the run_time_check=1 option, PowerMill will monitor the instantaneous current and power values during the dynamic simulation. In this case, PowerMill writes a message to the *.err* file each time the budget is violated, and a prints a summary of each violation in the *.power* file. You can use the **handle_ckt_error** command with the run_time_check= option to drop the simulation into the interactive mode whenever a budget violation occurs.

Finally, you can specify the check_dcpath=1 option, to trigger a DC path check each time an instantaneous current/power budget violation occurs.

Retrieving Power Information Interactively

You can interactively retrieve most power information, during the dynamic simulation, that is reported to the output files if you have previously instructed PowerMill to track that information.

You can use the following commands to obtain node current information:

get_node_i
get_node_iavg
get_node_irms
get_node_ipeak

These commands report the current value of the instantaneous, average, RMS, and peak currents, respectively. In each case, you must have requested the simulator to track the relevant information requested for the command to work (See "Analyzing Power at Full-Chip Level" and "Printing and Reporting Internal Node Currents" in this chapter).

You can use the following commands to obtain element and current information:

get_elem_i
get_elem_iavg
get_elem_irms
get_elem_ipeak

These commands report the current value of the instantaneous, average, RMS, and peak currents, respectively. In each case, the simulator reports the values for all element branches being tracked. You can track the current for an element branch using the commands discussed in "Printing and Reporting Branch Currents."

You can use the following commands to get interactive information on blocks created with **report_block_powr** command:

get_block_info
get_block_i
get_block_icont

The **get_block_info** command supplies a statistical description of the block, essentially identical to the header in the **report_block_powr** report, in the *.log* file. In addition, if the block was generated using the level= or at_level= options, it will show the block's hierarchical parent block, as well the number and names of its children.

The **get_block_i** command reports the instantaneous, average, and RMS values of all quantities tracked by the block.

The **get_block_icont** command generates hierarchical power information for blocks created using the **report_block_powr** level= or at_level options. This consists of a sorted list of the block's hierarchical children and their contributions to the specified currents.

EXAMPLE:

```
get_block_icont current=src type=inst xsr34x22.top
```

This command returns the contribution of each child block of the parent block 34x22 to the block's total supply current. See Figure 5 for a sample of the output of this command.

```
Block: xsr34x22.top
Total instantaneous source current          :-16054.265763 uA
Contributions from children -
        xsr34x22.x1158.top      : -7026.176334 uA
        xsr34x22.x1167.top      : -3376.436160 uA
        xsr34x22.x1166.top      : -2848.805135 uA
        xsr34x22.x1161.top      : -2100.318100 uA
        xsr34x22.x1169.top      : -691.675033 uA
        xsr34x22.x1164.top      : -7.744000 uA
        xsr34x22.x1160.top      : -5.689000 uA
        xsr34x22.x1163.top      :  4.136000 uA
        xsr34x22.x1162.top      : -2.858000 uA
        xsr34x22.x1170.top      :  1.320000 uA
        xsr34x22.x1165.top      :  0.000000 uA
        xsr34x22.x1168.top      :  0.000000 uA
        xsr34x22.x1159.top      : -0.000000 uA
        xsr34x22.x1157.top      : -0.000000 uA
```

Figure 5 Sample output from get_block_icont command

You can use the following interactive commands to get current information on individual probe nodes generated by the **report_block_powr** command, or the **assign_node_i** and **assign_branch_i** commands:

get_probe_i
get_probe_iavg
get_probe_irms
get_probe_ipeak

Controlling Power Reporting Resolution and Accuracy

The accuracy of the reported current and power values is dependent on the accuracy of the underlying simulation engine. Therefore, all commands that affect the accuracy of the timing or voltage also affect power. In addition, the following commands control different aspects of the simulation specific to power reporting:

set_sim_ires
set_print_ires
set_print_tres
set_power_acc

The **set_sim_ires** and **set_print_ires** commands control the current resolution of the current/power reporting. PowerMill ignores currents that are smaller than the values set by these commands.

The **set_print_ires** command affects only the current/power output and reporting—it does not affect the underlying simulation engine. In contrast, the **set_sim_ires** command does modify certain simulation parameters to ensure that the underlying simulation is done with sufficient accuracy to support the specified current resolution. In general, it is recommended that you use the **set_sim_ires** command instead of **set_print_ires** when you need to achieve a greater current resolution; otherwise, the reported current values might not be accurate. The default current resolution is 1 uA. The **set_print_ires** command typically does not impact performance, but does affect the size of the *.out* file. The **set_sim_ires** command can have a significant effect on the speed of the simulation.

The **set_print_tres** command controls the time resolution of all simulation output, printed to the *.out* file, as well as the time resolution used in calculating the reported average and RMS power numbers. The time resolution is set by default to be ten times the value of the simulation time resolution (see **set_sim_tres**). This setting should be adequate for most situations; however, if you want to get the maximum accuracy in the current calculation, you should set the print time resolution to be the same as the simulation time resolution. This setting will create a larger output file and possibly reduce the simulation performance since the output handles a larger number of points.

The **set_power_acc** command allows you to set the level of detail used in the accounting of circuit capacitances when doing element and node current calculations. Typically, the simplified models provide a high level of accuracy, and a significant improvement in simulation performance, when used to calculate the average and RMS current values. However, you might want to use the more accurate models if you need to get the detailed shape of the instantaneous waveforms. Specifically, you should apply the most accurate settings when comparing instantaneous waveforms of small circuits with another simulator, such as a SPICE simulation. Results for large circuits typically are not very sensitive to the capacitance model. The default settings usually provide a reasonable balance of accuracy and performance.

For more specific information on each command, see Chapter 2, “Configuration and Interactive Commands” in the *PowerMill Reference Guide.*

Printing and Reporting Branch Currents

Using PowerMill, you can print many different types of branch currents. There are commands that print instantaneous, average, and RMS currents as waveforms to the *.out* file. PowerMill also provides commands to report average, RMS, and peak current summaries to the *.log* file

The branch current printing commands follow the HSPICE conventions for i1, i2, i3, and i4 currents. The following list shows the branch current mapping for MOS transistors.

i1 = drain current
i2 = gate current

i3 = source current
i4 = bulk current

PowerMill also provides commands for printing and reporting DC current for specified elements. Use the following commands to print all types of branch currents.

print_branch_didt1
print_branch_didt2
print_branch_didt3
print_branch_didt4
print_branch_didtdc
print_branch_i1
print_branch_i1avg
print_branch_i1rms
print_branch_i2
print_branch_i2avg
print_branch_i2rms
print_branch_i3
print_branch_i3avg
print_branch_i3rms
print_branch_i4
print_branch_i4avg
print_branch_i4rms
print_branch_idc
print_branch_idcavg
print_branch_idcrms
report_branch_i
report_branch_powr

For more specific information on each command, see Chapter 2, “Configuration and Interactive Commands” in the *PowerMill Reference Guide.*

Printing and Reporting Internal Node Currents

It is sometimes useful to be able to measure the amount of current flowing through an internal (non-voltage source) node. Since, according to Kirchhoff’s Current Law, the algebraic sum of all currents at a node is zero, internal nodes must be treated differently from voltage source nodes. For this reason, PowerMill defines the internal node current to be the sum of the positive currents flowing into the node.

Use the following commands to print and report internal node currents (see "Analyzing Power at Full-Chip Level" on page 86*)*.

print_node_i	**print_node_didt**
print_node_iavg	**report_node_i**
print_node_irms	**report_node_powr**

For more specific information on each command, see the *PowerMill Reference Guide*.

Working with Probes

Under some circumstances, the standard block-level power reporting commands (see "Analyzing Power Block-by-Block" on page 87 and "Measuring Wasted Power" on page 88) might not provide sufficient flexibility. For these cases, PowerMill provides the commands **assign_node_i** and **assign_branch_i**. Together these commands allow you to construct a completely arbitrary sum of element branch currents.

EXAMPLE 1:

```
assign_branch_i drain d_sum mos1 mos2 mos3 mos4
```

This example constructs a sum of drain current for several MOS transistors and assigns it to the `d_sum` probe.

EXAMPLE 2:

```
assign_node_i any_node n_sum mos1 mos2 mos3 mos4
```

This example obtains the sum of all currents from these transistors into the node `any_node`.

The same probe name can be used in multiple "assign" statements to build up the desired sum.

Once the probe is constructed, the commands **print_probe_i** and **report_probe_i** can be used to obtain *.out* file waveforms and *.log* file current summaries for the probe identical to those created for the node currents. In addition, the **report_node_powr** command requests current information for all probe nodes if used without specifying any nodes. These commands can also be used with the **report_block_powr** command to obtain waveform information from those commands.

Current Histograms and Tracking Windows

PowerMill can generate a current histogram for each node, branch, and probe. The current histogram report consists of a report of the average, RMS, and peak currents over a set of user-defined intervals printed to the *.hist* file. A sample current histogram report for nodes VDD and GND is shown in Figure 6.

You can print a histogram using the **report_node_i**, **report_elem_exi**, and **report_probe_i** configuration commands. Alternatively, you can use the **report_ckt_phist** command, which automatically generates current histograms for each constant voltage source node, as well as all user-defined probe nodes. This automatically generates histograms for all power nodes including probes created by **report_block_powr**.

```
Node: XGND

  Start (ns)    Stop (ns)     Avg (uA)     RMS (uA)     Peak (uA)    Time (ns)
  5.0000e+01   9.0000e+01   2.8779e+01   5.3431e+01   1.0100e+02   6.0400e+01
  9.0000e+01   1.3000e+02   7.0781e+01   8.4014e+01   1.1700e+02   1.0245e+02
  1.3000e+02   1.7000e+02   5.3766e+01   7.3174e+01   1.0100e+02   1.5040e+02
  1.7000e+02   2.1000e+02   2.1036e+01   4.5936e+01   1.3500e+02   2.0232e+02
  2.1000e+02   2.5000e+02   1.0000e+02   1.0000e+02   1.0000e+02   2.1000e+02
  2.5000e+02   2.9000e+02   9.0790e+01   9.0970e+01   1.0100e+02   2.5040e+02
  2.9000e+02   3.3000e+02   7.3625e+01   7.3732e+01   8.2000e+01   2.9000e+02
  3.3000e+02   3.7000e+02   6.1875e+01   6.1934e+01   6.8000e+01   3.3000e+02
  3.7000e+02   4.1000e+02   5.4000e+01   5.4032e+01   5.8000e+01   3.7000e+02
  4.1000e+02   4.5000e+02   4.9000e+01   4.9015e+01   5.1000e+01   4.1000e+02


Element: IOBUFF0.P1    Branch: 1

  Start (ns)    Stop (ns)     Avg (uA)     RMS (uA)     Peak (uA)    Time (ns)
  5.0000e+01   9.0000e+01   1.0304e+03   4.9851e+03   3.3619e+04   6.2000e+01
  9.0000e+01   1.3000e+02   1.4845e+03   7.0174e+03   5.1510e+04   1.0190e+02
  1.3000e+02   1.7000e+02   1.0614e+03   4.9601e+03   3.3150e+04   1.5200e+02
  1.7000e+02   2.1000e+02   8.5798e+02   5.1421e+03   4.3718e+04   2.0181e+02
  2.1000e+02   2.5000e+02   1.2009e+02   1.2009e+02   1.2100e+02   2.1000e+02
  2.5000e+02   2.9000e+02   2.9633e+01   5.0674e+01   3.2600e+02   2.5214e+02
  2.9000e+02   3.3000e+02   1.1750e+01   1.1874e+01   1.5000e+01   2.9000e+02
  3.3000e+02   3.7000e+02   7.2500e+00   7.3144e+00   9.0000e+00   3.3000e+02
  3.7000e+02   4.1000e+02   4.6250e+00   4.6771e+00   6.0000e+00   3.7000e+02
  4.1000e+02   4.5000e+02   2.7500e+00   2.7839e+00   4.0000e+00   4.1000e+02
```

Figure 6 Sample current histogram report

You can specify the histogram intervals using the **set_print_iwindow** command.

EXAMPLE 1:

```
set_print_iwindow period=100ns start=25n end=238ns
   on_time=30ns
```

This example specifies the following current windows: 1) from 25 ns to 55 ns; 2) from 125 ns to 155 ns; 3) from 225 ns to 238 ns.

When a set of current tracking intervals is created by the **set_print_iwindow** command, the simulation only tracks current during the specified intervals. Therefore, waveforms printed to the *.out* file are limited in time to the specified intervals. Also, the average, RMS, and peak currents and waveforms are only computed based on the periods when tracking is enabled. This can be useful when you want to ignore part of the simulation run when calculating the current statistics.

Using the GAP Feature to Estimate Maximum Power

PowerMill's GAP (Genetic Algorithm for maximum Power estimation) feature searches input vector pairs or triplets to maximize the power dissipation of a given circuit. The GAP feature combines a genetic algorithm with advanced circuit simulation technology to efficiently generate a tight lower bound on the maximum power estimation of a given circuit. The power computed is the summation of the power dissipated and the power stored in the circuit.

A genetic algorithm is a general-purpose optimization algorithm. Through a mechanism analogous to natural selection, it can search the solution space to minimize/maximize a specified objective function, called a "fitness" function. Part of this work is based on the Synopsys-sponsored university research project led by Professor Kwang-Ting Cheng and Ph.D. Candidate Yi-Min Jiang from the Department of Electrical and Computer Engineering at the University of California, Santa Barbara.

The GAP feature is able to maximize four objective functions:

- Average power per clock cycle of the circuit
- Instantaneous power of the circuit
- Average value per clock cycle of the customized fitness function
- Instantaneous value of the customized fitness function

The GAP feature generates a tight lower bound on the maximum power of a given circuit. At the beginning of the estimation, GAP selects a specific group of vector sets to be kept. These vector sets are comprised of either vector pairs or triplets, depending on the circuit type (see tutorials 5, 6, and 7 in Chapter 3 for details). The GAP feature measures the quality of each vector pair based on the power consumption it produces when applied to the circuit. At each generation, GAP performs operations on the vector files (in the kept group) to improve their quality. The concept of the quality of vector pairs improving with every generation is analogous to the biological concept of natural selection.

You can control the size of the specific group, called the population size, using the p_size= option of the **guide_gap_search** configuration command. In general, a larger population size generates higher-quality vector sets, while increasing the CPU time needed for the estimation.

You can use the GAP result when analyzing various reliability issues. For example, if the lower bound of maximum power, generated by GAP, exceeds your power budget, you will probably need to redesign the block.

The following PowerMill configuration and interactive commands are part of the GAP feature for estimating maximum power consumption:

guide_gap_search	**set_gap_opt**
report_gap_prediction	**report_gap_parameters**
report_gap_progress	**use_vec_gap**

Running a Basic GAP Estimation

This section provides a general procedure for using the GAP commands. It is strongly recommended that you run the following tutorials in Chapter 3, “PowerMill Tutorials” before running the GAP feature:

- “Tutorial 5: Estimating Maximum Power for Combinational Circuits” on page 41

- "Tutorial 6: Estimating Maximum Power for Sequential Circuits" on page 50
- "Tutorial 7: Customizing a GAP Objective Function using an ADFMI Code File" on page 69

NOTE: Searching for the optimum pairs or triplets to maximize power consumption is a time-consuming optimization process—even for small circuits. For example, it might take about 10 minutes of CPU time to estimate a small sequential circuit with approximately 200 CMOS transistors. In general, the GAP estimation is more efficient, in results and CPU time, than a random simulation.

Procedure

1 Use the configuration command **use_vec_gap** to specify the vector file (with type "vec") to be used for the maximum power estimation. For sequential circuits, you can only provide the primary input vector file. You can automatically generate a vector file containing all the pseudo-primary inputs (PPIs) by applying the **mark_node_latch** command. See "Tutorial 6: Estimating Maximum Power for Sequential Circuits" on page 50 a detailed description of PPI detection.

In the vector file, inputs, biputs, and pseudo-primary inputs (output of memory elements) should be distinguished by character "i", "b", and "p". If no io command is specified, all the signals are considered to be inputs. You can specify one or more vector files. The following sample configuration file shows how multiple vector files are specified:

```
use_vec_gap vec=sep1.vec vec=sep2.vec
set_gap_opt period=10ns vec_history=3
guide_gap_search time=30 pattern=20000 fit=2
```

The vector files can contain vectors to be used by the genetic algorithm as part of the first generation.

2 Use the **set_gap_opt** command to set various GAP options for controlling the estimation flow. This command has two parameters that you can set:

- Use the period= option to set the clock period to be used by the GAP estimation. For sequential circuits, the period should be the same as the clock signal defined in the netlist file (or *.cmd* file). If you do not specify a period, the GAP estimation will use the default period of 20 ns.
- Use the delay= option to synchronize GAP with the signal clock (if it is not started from time 0).
- Use the vec_history= option specify which vectors will be kept in the history files. At the end of the simulation, GAP produces a history file corresponding to each vector file used during the estimation. If you do not specify a history option, GAP keeps only the vector pairs or triplets from the generation producing the maximum power.
- Use the save_history= option to save the circuit state during a GAP simulation. You can use the save_history option in conjunction with the vec_history option to specify how the state files of the circuit will be saved. You can use the saved state files to either resume the GAP estimation from a saved time, or do advanced power analysis for specific generations under GAP.

  The range of the saved state information is dependent on the setting for the vec_history= option:

  vec_history = 1 saves only the state at the beginning of the generation that produces the maximum power.

  vec_history = 2 saves the state at the beginning of all the generations that improve the maximum power estimation.

vec_history = 3 saves the state at the beginning of all generations.

The save_history= option is disabled by default.

3 Use the **guide_gap_search** command to specify additional parameters for the GAP estimation. You can use this command to set the following parameters:

- Use the level= option to select the search level to be applied to the GAP estimation. The search level is an integer between 1 and 10 that defines a set of parameters to be applied to the genetic algorithm. A higher search level generally produces a better result, but requires more CPU time. The default value is 5.
- Use the fit= option to specify the objective to be maximized (that is, the fitness function). Currently, the GAP feature supports two objective functions: *instantaneous* power (fit=2) and *average* power per clock cycle (fit=1, default).
- Use the max_vec= option to limit the number of applied input vector pairs or triplets during the estimation. The estimation stops when it reaches this limit. The default value is positive infinity.
- Use the max_value= option to limit the value of the fitness function. The estimation stops if the value of the fitness function exceeds this threshold. This value uses MKS units and has a default value of positive infinity.
- Use the max_gen= option to limit the number of generations applied. The default value for this option is positive infinity.
- Use the time= option to limit the CPU time used by the simulation. The simulation stops when it reaches this limit.

- Use the p_size= option to specify the population size used by the genetic algorithm. Acceptable values for population size are any integers between 20 and 50. A larger population size tends to generate a better (larger) estimate with a longer CPU time. The default value is 20.

4. During the estimation, you can use the **report_gap_progress** interactive command to retrieve information on the progress of the simulation and display it in either a table or a graph. At the end of the GAP estimation, PowerMill automatically generates a *.gap* summary file and one or more history files (*.gav*). The input vectors used during the simulation are kept in the history files, which can then be used as normal vector files in another simulation (using the -nvec PowerMill command-line option).

5. You can use the **report_gap_parameters** interactive command to show the current settings for all of the GAP commands.

Creating a Customized Fitness Function

For a GAP estimation, you can specify one of four types of fitness functions using the fit= option of the **guide_gap_search** command. Types 1 and 2 are based on the peak average power per clock cycle and peak instantaneous power, respectively, of the whole circuit. These functions are built into the software. In addition, you can use types 3 and 4 to apply a customized objective function defined in an ADFMI C file.

To create a customized objective function:

1. Enable GAP customization by specifying the fit=3 or fit=4 option with the **guide_gap_search** configuration command. Fit=3 applies a function based on the peak average value per clock cycle while fit=4 applies a function based on peak instantaneous value.

2. In an ADFMI C file, create a regular C function with type double and void argument.

The GAP estimation will use the value returned by this C function as the objective to be maximized. The following table lists the types of arguments you can use to define the fitness measure in your C function:

Number	Type	Argument Name
1	Node Voltage	v(*node_name*)
2	Node Logic	l(*node_name*)
3	Node Current	inode(*node_name*)
4	Pin Current	ipin(*pin_name*)
5	Branch Current	i*branch_num*(*elem_name*)
6	Block Power	p(*block_label*)
7	Probe Current	iprobe(*probe_name*)
8	Math Signal	m(*signal_label*)

EXAMPLE:

```
fmRegisterGap("l(cin1) ipin(x11.ain) v(cin1)
   p(x11)m(dot)", fitness);
```

This example specifies that a total of five arguments (`l(cin1)`, `ipin(x11.ain)`, `v(cin1)`, `p(x11)`, and `m(dot)`) to be used in the customized objective function for PowerMill's GAP feature. It also specifies "fitness" as the function pointer to the customized objective function (defined in the ADFMI C file).

You can access the value of an argument either by name using the **fmGapGetValueByName()** API function, or by ID using the **fmGapGetValueByID()** function (see the *ADFMI Manual* for more information). If you are using arguments of type 6 or 7, you first need to create the corresponding blocks or probes using the appropriate configuration commands (for example, **report_block_powr**, **assign_node_i**, etc.). Likewise, to use arguments of type 8, you need to define the math signals using HSPICE .print, .probe, or .plot control lines.

3 Register the objective (fitness) function.

In the ADFMI registration function (**RegisterUserModel()**), invoke the GAP registration function **fmRegisterUserGap()** (see Figure 7).To it you need to provide a) the function pointer of the objective C function, and b) a string containing the names of all the arguments used in defining the fitness measure. In the string, the space character serves as the delimiter to separate argument names.

```
  * Customized objective function to be maximized
  */
double fitness()
{
  static int id_4_block1_powr, id_4_block2_powr;
  double sum = 0;

  if (fmSimPhase() == FMDCINITPHASE) {
    /* get id for signals, which can be used later to
       access their value */
    id_4_block1_powr = fmGapGetSignalId ("p(block1)");
   id_4_block2_powr = fmGapGetSignalId ("p(block2)");
    return 0;
  }
  else {
    /* calculate fitness measure */
    sum =
fmGapGetValueById(id_4_block1_powr)+fmGapGetValueById
(id_4_block2_powr);
    return sum;
  }
}

void
RegisterUserModel()
{
  /* Specify block power for block1 and block2 */
 fmConfigCmd ("report_block_powr block1 track_power=1
x1*");
 fmConfigCmd ("report_block_powr block2 track_power=1
x2*");

  /* Register arguments and function pointer of
customized objective function */
  fmRegisterGap("p(block1) p(block2)", fitness);
}
/*
```

Figure 7 Sample custoomized objective function

This example uses **report_block_powr** configuration command to create two block power signals, which it then uses to define the customized fitness function.

See the following tutorials in Chapter 3, “PowerMill Tutorials” for more information on using the GAP feature:

- “Tutorial 5: Estimating Maximum Power for Combinational Circuits” on page 41
- “Tutorial 6: Estimating Maximum Power for Sequential Circuits” on page 50
- “Tutorial 7: Customizing a GAP Objective Function using an ADFMI Code File” on page 69

Performing a Dynamic Power Consumption Analysis

This section provides an overview of the PowerMill features that can be used to identify and diagnose, during the dynamic simulation, conditions in your circuits that could lead to excessive power consumption.

Information is included on the following subjects:

- Dynamic Rise/Fall Time Checking
- Detecting Dynamic Floating Nodes (Z-State Nodes)
- Detecting Excessive Branch Currents
- Analyzing Hot-Spot Node Currents
- Detecting Dynamic DC Paths

Checking Dynamic Rise and Fall Times (U-State Nodes)

There are a couple of reasons why a circuit could end up consuming an unexpectedly large amount of power. If a node spends an excessive amount of time in a U-state, it could create a large short-circuit current.

The **report_node_u** and **report_node_quick** configuration commands provide a mechanism to check for excessively long and excessively short rise/fall times, respectively, during the dynamic simulation. Each command checks the time taken by the node to transition between its logic high and low threshold and reports violations to the *.err* file. By default, the logic high and low thresholds are set to 70% and 30% of the nodes high voltage, respectively. You can use the **set_node_thresh** command to adjust the default voltage.

Detecting Dynamic Floating Nodes (Z-State Nodes)

If a node is left floating (Z-state) for an excessive time, the node voltage could drift into an intermediate voltage level due to sub-threshold short-circuit currents in the associated elements. This can lead to unexpected DC paths in logic blocks that are in the node fanout.

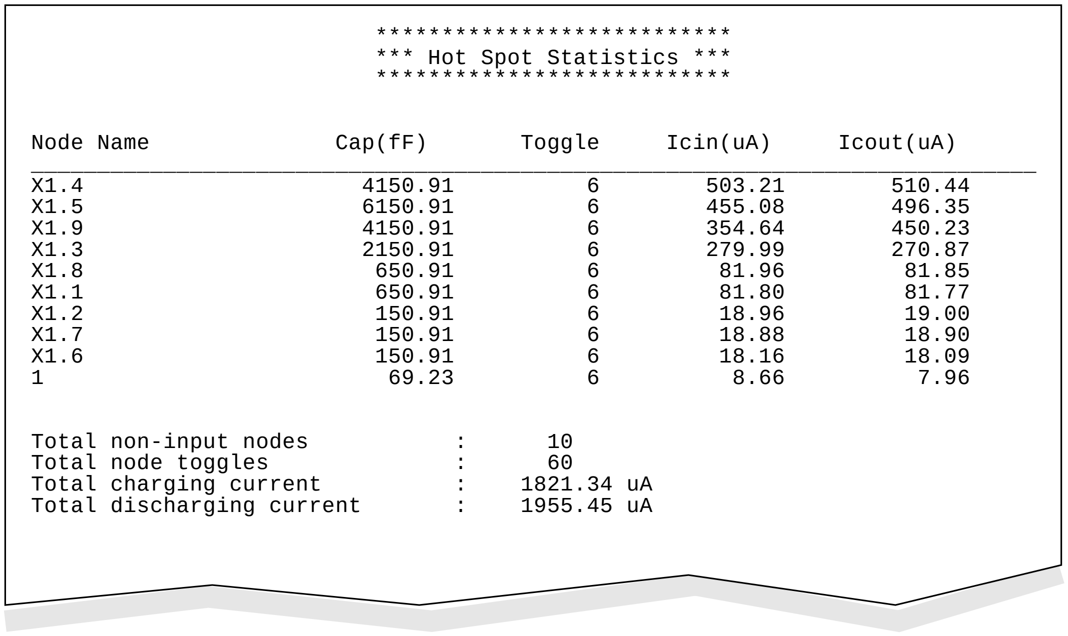
The **report_node_z** configuration command provides a mechanism to report any node that stays in the Z-state longer than a user specified time during the dynamic simulation. Violations are reported to the *.err* file.

Detecting Excessive Branch Currents

When attempting to diagnose excessive power in a circuit, it is often helpful to determine which circuit elements might be carrying unexpectedly large currents. You can use the **report_elem_exi** command monitor the currents of specified elements and report those elements that have currents exceeding a specified threshold for a specified time during the dynamic simulation. PowerMill reports violations to the *.err* file.

Analyzing Hot-Spot Node Currents

When you are redesigning for lower-power consumption, it's helpful to know how each circuit node contributes to the total power consumption. The **report_node_hotspot** command lists the average current statistics for the specified nodes in the *.sum* file. A sample hot spot report is shown in Figure 5.



```
                            **************************
                            *** Hot Spot Statistics ***
                            **************************


Node Name                 Cap(fF)       Toggle     Icin(uA)     Icout(uA)
____________________________________________________________________________
X1.4                        4150.91          6        503.21        510.44
X1.5                        6150.91          6        455.08        496.35
X1.9                        4150.91          6        354.64        450.23
X1.3                        2150.91          6        279.99        270.87
X1.8                         650.91          6         81.96         81.85
X1.1                         650.91          6         81.80         81.77
X1.2                         150.91          6         18.96         19.00
X1.7                         150.91          6         18.88         18.90
X1.6                         150.91          6         18.16         18.09
1                             69.23          6          8.66          7.96


Total non-input nodes           :      10
Total node toggles              :      60
Total charging current          :    1821.34 uA
Total discharging current       :    1955.45 uA
```

Figure 8 Sample hot spot report

The hot spot report provides the following information for each node:

- **Average Node Capacitance**—lumped from netlist, wiring, and transistor gate and diffusion capacitances.
- **Toggle Count**—total number of logic toggles for the node.
- **Average Charging Current**—average current flowing into the capacitances connected to the node.
- **Average Discharge Current**—average current flowing out of the capacitances connected to the node.

If a node toggle count is as expected, the charge and discharge currents represent the mandatory current consumption to support the necessary circuit operation.

Ideally, the product of the node capacitance, toggle count, and supply voltage will equal the sum of the charging and discharging current multiplied by the simulation period. Thus, if

```
(toggle count)*(node capacitance)*(Vdd) - (charge +
discharge)*(simulation time)
```

is greater than 0, some toggles cross the logic threshold, but never reach the full-swing value (either VDD or 0). Conversely, if the difference is less than 0, the node voltage waveform could show some glitches, either above or below the logic threshold voltage for which no toggle is counted.

A large circuit with a large number of nodes could produce a huge volume of data in the *.sum* file, which would be too complicated to analyze. You can use the **set_hotspot_factor** command to suppress those nodes for which the sum of the charging and discharging current is less than a specified factor multiplied by the sum for the node with the largest currents.

The report in Figure 8 can be used to illustrate the effect of applying a hot spot factor of 0.5. As you can see, the node with the largest currents is `X1.4`, which has a charging current of `503.21` µA and a discharging current of `510.44` µA. Applying the 0.5 hotspot factor, the sum of the charging and discharging currents multiplied by the hotspot factor equals 506.83 µA. The sum of any other node's charging and discharging current would have to be greater than this number to be included in the individual node report. In this case, all nodes below `X1.3` are excluded from the report.

The summation of charge currents for all nodes within a block represents the average charge current for the block. Conversely, the summation of discharge currents for all node within a block represents the average discharge current for the block. The difference between the total average block current and the charge current should be the average short-circuit current from DC short-circuit, or the short-circuit short-circuit current, during the transition period when both NMOS and PMOS transistors were conducting. This is one way of estimating the circuit short-circuit current.

Detecting Dynamic DC Paths

In CMOS circuits, any DC current flowing in the circuit between power and ground is considered to be wasted power. Therefore, wasted power can come from “crowbar” or “short-circuit” current caused by both P-channel and N-channel devices being on during a transition. Wasted power can come from mismatched supply voltages, a failure to reach full swing, timing errors, or other unintended sources. It can also come from unavoidable analog bias circuits and current mirrors in operational amplifiers.

The **report_ckt_dcpath** command provides a method of detecting DC paths that occur during the dynamic simulation. You can use the **set_dcpath_thresh** command to prevent the reporting of DC paths with small currents. A DC path is reported only if the DC current through each element in the path exceeds the threshold. You can use this to exclude paths for bias circuits from the report, if they are sufficiently small. Alternatively, you can use the **limit_dcpath_search** command to exclude certain blocks from the search or to limit the search to specific blocks.

The detected DC paths are reported to the screen and to the *.dcpath* file. A sample DC path report is shown in Figure 9.

```
Nonzero dc path(s) found at time 200 ns

Path # 1
-----------------------------------------------------------------------------
      resistor IOBUFF0.R1                              current =  -0.02 uA
          from XGND                                    voltage =   0.00 V
            to DATAOUT0                                voltage =   0.00 V
      resistor IOBUFF0.R2                              current =  79.98 uA
          from IOBUFF0.VR                              voltage =   4.00 V
            to DATAOUT0                                voltage =   0.00 V
-----------------------------------------------------------------------------

Path # 2
-----------------------------------------------------------------------------
      resistor IOBUFF0.R1                              current =  -99.97 uA
          from XGND                                    voltage =   0.00 V
            to DATAOUT0                                voltage =   5.00 V
      inductor IOBUFF0.OUTBOND                         current =  -119.93 uA
          from DATAOUT0                                voltage =   5.00 V
            to IOBUFF0.OUT1                            voltage =   5.00 V
    transistor IOBUFF0.P1                  (type PMOS) current =  -119.91 uA
          gate IOBUFF0.INGATE                          voltage =   0.00 V
         drain IOBUFF0.OUT1                            voltage =   5.00 V
        source IOBUFF0.DP1                             voltage =   5.00 V
    transistor IOBUFF0.P2                  (type PMOS) current =  -119.93 uA
          gate IOBUFF0.PGATE                           voltage =   0.00 V
         drain IOBUFF0.DP1                             voltage =   5.00 V
        source EVDD                                    voltage =   5.00 V
      inductor IOBUFF0.L1                              current =  -119.93 uA
          from EVDD                                    voltage =   5.00 V
            to VDD                                     voltage =   5.00 V
-----------------------------------------------------------------------------
```

Figure 9 Sample DC path report

Performing a Static Power Consumption Diagnosis

This section provides an overview of the static power diagnosis features available in PowerMill. You can use these features to analyze your circuit based on its topology and look for conditions that might lead to excessive power use. They are intended to provide an alternative to the dynamic power diagnosis when dynamic simulation requires too much simulation time and you want to scale down power consumption. These analyses are performed before the dynamic simulation begins and generally require significantly less simulation time than a full-dynamic simulation.

This section includes information on the following subjects:

- Detecting Static DC Paths
- Static Excessive Rise/Fall Time Detection

Detecting Static DC Paths

A static DC short-circuit path exists if a node has a static conducting path to VDD and a static conducting path to GND. Each node in both paths are in a fighting condition at all times. Use the **report_ckt_leak** command to flag such nodes.

However, a static DC short-circuit path, which is independent of the external stimulus, rarely exists in CMOS circuits. A dynamic short-circuit path is more likely, however, it requires a specific input condition that might not occur. The **report_ckt_leak** command can find a static DC path, as well as detect several unusual topological conditions in regular CMOS designs. These conditions, although insufficient to confirm a definite DC short-circuit path, might uncover potential short-circuit paths.

The **report_ckt_leak** command searches for the following conditions:

- **Partial Swing Detection**—A node is driven by NMOS (PMOS) transistors only and it drives at least one PMOS (NMOS) transistor's gate.
- **No Path to VDD/GND**—There is no possible conducting path to the node from a supply (ground) node.
- **Static Path to VDD/GND**—There exists a static conducting path from a supply (ground) node to the node.
- **Node is Stuck at High/low Voltage**—A static conducting path exists between the node and a supply (ground) node. In

addition, there is no possible conducting path between the node and any ground (supply) node.

- **Static DC Path**—There exists a static conducting path between two voltage source nodes.

For the conditions mentioned above, the **report_ckt_leak** command does not take into account difficulties due to multiple supplies or grounds with different voltages. All supplies (constant voltage sources with voltages above the logic high threshold) and grounds (constant voltage sources below the logic high threshold) are assumed equivalent.

For circuits that have multiple supplies or grounds, **report_ckt_leak** also searches for the following conditions, which can lead to unintended DC paths:

- **Partial off PMOS**

  A PMOS transistor with a source or drain that has a possible channel-connected path to a supply or input node with high voltage VDDH. In addition, its gate has a possible channel-connected path to a supply or input node with high voltage VDDL, where the voltage of VDDH is greater than VDDL.

- **Partial off NMOS**

  An NMOS transistor with a source or drain that has a possible channel-connect path to a ground or input node with low voltage GNDL. In addition, its gate has a possible channel-connect path to ground or input node with low voltage GNDH, where the voltage of GNDH is greater than GNDL.

- **Forward bias PMOS**

  A PMOS transistor with a source or drain terminal that has a possible channel-connected path to a supply or input node with high voltage VDDH. In addition, its substrate is connected to a voltage source of voltage VDDL, where VDDH is greater than VDD.

- **Forward bias NMOS**

An NMOS transistor with a source or drain terminal that has a possible channel-connected path to a ground or input node with high voltage GNDL. In addition, its substrate is connected to a voltage source of voltage GNDH, where GNDH is greater than GNDL.

PowerMill reports any problems it detects to the *.err* file. For details of the syntax of these reports see Chapter 7, “Utilities for Dynamic EPIC Tools” in the *EPIC Tools Reference Guide*.

Static Excessive Rise/Fall Time Detection

A large transient short-circuit current might occur if the voltage rise/fall transition time for a node is excessive. You can use the **report_node_maxrf** command to report nodes with worst-case transition times that exceed a specified threshold time.

The node rise/fall time is the time taken for the node to make the transition between its high and low logic thresholds. This time is estimated based on the lumped capacitance for the node and the maximum impedance driving it. By default, the logic high and low thresholds are set to 70% and 30%, respectively, of the node’s high voltage. You can adjust these default levels using the **set_node_thresh** command.

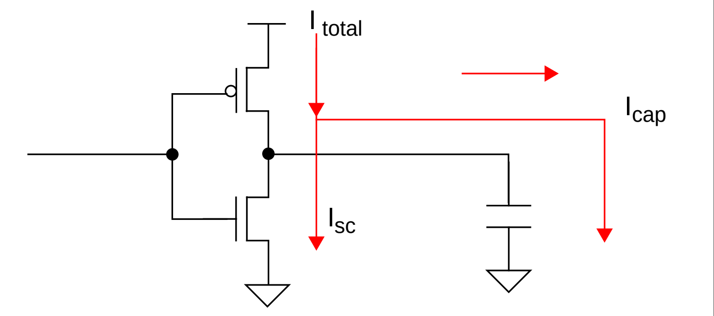
PowerMill reports any problems it detects to the *.err* file. For details on the syntax of these reports, see Chapter 7, “Utilities for Dynamic EPIC Tools” in the *EPIC Tools Reference Guide*.

Analyzing Power Using the RC and UD Simulation Modes

The RC and UD modes provide a fast power estimation for digital CMOS circuits. In most digital circuits, major power contributions come from switching activities. Therefore, the RC and UD modes—which use switch-level simulation techniques—produce a faster average power estimation with reasonable accuracy, as compared to the default PWL mode. RC mode provides full-delay information for CMOS circuits, while UD mode only provides fixed unit-delay information.

Technical Background

The power dissipation in digital CMOS circuits consists of two major parts: capacitive power and short-circuit power (see Figure 10).



$$\text{percent short-circuit} = \frac{I_{sc}}{I_{total}} \times 100$$

$$I_{total} = I_{cap} + I_{sc}$$

Figure 10 Short-circuit (sc) current calculation

Switching current usually plays a major role in most digital circuits. When running in RC or UD mode, the simulator calculates the average power based on the charging and discharging of node capacitances. That is, the charging or discharging current due to a voltage change (dV) at a node is `C *(dV/dt)`, where `C` is the capacitance and `dt` is the transition time of the node. This can accurately estimate the average switching power in circuits.

For CMOS circuits, short-circuit current estimation in the RC or UD mode is based on the input and output slopes and is less accurate than PWL mode. In RC or UD mode, the simulator estimates the short-circuit current (I_{sc}) using the switching capacitive current (I_{cap}) and the ratio of the input slope and output slope. There will always be some discrepancy between the estimated short-circuit current calculated in the RC and UD modes and the actual short-circuit current calculated in PWL mode. If the short-circuit current is small, this discrepancy is negligible. However, if a circuit contains a large short-circuit current (if the input slope is much slower than the output slope) the discrepancy between RC and UD mode results and PWL mode results could increase significantly. In this case, a power analysis using RC or UD mode would be inaccurate.

Another source of discrepancy of average power—between PWL and RC modes—is derived from the application of different timing models, which could cause new glitches or existing glitches to change width and height or even disappear.

Running a Simulation in RC or UD Mode

You can instruct the simulator to run in RC mode in two places. You can either use the `-d rc` command-line option when starting the simulator or you can use the use `set_sim_mode rc` command in the configuration file. Likewise, you can use the `-d ud` command-line option or the `set_sim_mode ud` command to run in the UD power analysis mode.

Power analysis in RC and UD mode is based on switching capacitive power in each stage, which generally corresponds to a gate. Therefore, only the average power of a stage is meaningful and, as a result, the instantaneous power is displayed based on the average power. You can combine RC and UD stages with PWL stages to achieve a balance between speed and accuracy. PowerMill’s RC and UD modes apply a few basic circuit detection rules to the circuits and mark portions of the circuit to use the PWL mode.

You can apply the following power diagnosis commands to RC and UD stages:

report_block_powr
Use this command to analyze power use for specified stages. If the specified elements cover only a partial stage, this command forces the whole stage into PWL mode.

EXAMPLE 1:

```
report_block_powr blk2 xadder.* blk1
```

This command reports the average power dissipation for a circuit block, in this case, an adder circuit.

report_probe_i and **print_probe_i**
Use these commands to report or print current information for probes created by the **report_block_powr** command.

EXAMPLE 2:

```
print_probe_i avg blk1_vdd
print_probe_i avg total_src[blk2]
```

This example prints the average VDD current waveform of the adder block specified in Example 1.

report_node_i and **print_node_i**
Use these commands to get the total power dissipation of RC and UD stages by applying them to power nodes (VDD/GND) connected to these stages. Applying them to non-power nodes forces the stages into PWL mode.

EXAMPLE 3:

```
report_node_i avg VDD
report_block_powr blk1 track_wasted=1 *
report_block_powr blk2 *
```

These commands report the power dissipation for the whole circuit, assuming VDD is the only power source.

CAUTION: Applying **report_branch_i**, **print_branch_i***, **print_node_v**, **report_elem_exi**, **assign_branch_i**, or **assign_node_i** causes the given elements to be PWL, which defeats the fast power estimation feature for elements intended to be RC or UD elements.

Chapter 5

Using the ACE Feature

Overview

The Analog Circuit Engine (ACE) is a built-in feature that extends PowerMill's capability to simulate analog and mixed-signal circuit designs.

The default autodetection algorithms can simulate most analog CMOS circuits (such as bandgap references and PLLs) in mixed-signal designs with no additional configuration commands. If you are simulating sophisticated or complex designs, you might want to apply some additional configuration commands to improve the accuracy of a simulation.

Since output printing and other controls are usually included in the SPICE netlist, you will most likely not need a configuration file when running ACE.

Included in this Chapter

This chapter highlights some of the capabilities of the ACE option. The following sections are included:

- Selecting Autodetection Rules
- Controlling Node Sensitivity
- Controlling the Simulation Time Resolution
- Applying Multiple Time Steps (Multi-Rate Simulations)
- Controlling Waveform Print Resolution
- Applying Multiple Simulation Modes
- Simulating BJTs

Selecting Autodetection Rules

The ACE option provides a group of autodetection commands you can use to create a simulation that is balanced in terms of accuracy and performance. It includes a default set of rules as well as several sets of rules that can be applied as needed to improve either the accuracy or speed of a simulation.

The default autodetection rules, controlled by the **search_ckt_msx** command, provide acceptable accuracy and speed for most mixed-signal circuits such as PLLs, ADCs, DACs, switched-capacitors, etc. However, some circuits might require a more accurate simulation, in which case, you could apply the **search_ckt_analog** command. This command applies a more extensive set of rules and is more accurate. However, a simulation using **search_ckt_analog** command will run somewhat slower as compared to a simulation using the default **search_ck_msx** rules. The actual effect on speed will depend on the type of circuit, the amount searched, and how much the **search_ckt_analog** rule changes settings from the **search_ckt_msx** rule.

You can use the following configuration commands to apply different sets of autodetection rules, as needed:

- **search_ckt_msx** (default)

  Use this set of rules for mixed-signal circuits. This is the default rule setting, except when running in RC or UD simulation modes. It includes most of rules used by **search_ckt_analog**, but is less conservative and runs faster.

- **search_ckt_analog**

  Use this set of rules for analog circuits. This command applies a more conservative autodetection algorithm.

- **search_ckt_mem**

  Use this set of rules for embedded DRAMs, PLDs, and ROMs. You can also use the default **search_ckt_msx** for these circuits, but the simulation will likely take longer to run.

- **search_ckt_cpump**

  Use this set of rules for charge-pump circuits, flash memory circuits, etc. Flash memory circuits can usually be simulated with the default **search_ckt_msx** set of detection rules; however, some circuits might require the more conservative **search_ckt_cpump** rules.

- **search_ckt_logic**

Use this set of rules for digital circuits. This set of rules is more aggressive and can greatly improve the simulation speed for digital circuits.

- **search_ckt_rc**

  Use this set of rules for digital-only blocks. If you use the set_sim_mode rc command, this set of rules is applied to the entire circuit by default.

- **search_ckt_ud**

  Use this set of rules for digital-only blocks. If you use the `set_sim_mode ud` command, this set of rules is applied to the entire circuit by default.

Applying Multiple Autodetection Rules

The ACE option allows you to apply different autodetection rules to individual portions of a circuit. If you do not specify an autodetection command, by default, the **search_ckt_msx** rules are applied to the entire circuit. However, if you apply an autodetection command to a portion (a block) of the circuit, that portion is searched with rules set by that autodetection command (that is, the default **search_ckt_msx** is not applied to that portion of the circuit). If you apply more than one autodetection command to the same block in a circuit, the simulator combines the rules for both commands and applies them to the specified block.

EXAMPLE 1:

```
search_ckt_mem el=xdram.*
```

This command applies `search_ckt_mem` rules to the `xdram` block. The default **search_ckt_msx** is applied to the remainder of the circuit.

EXAMPLE 2:

```
search_ckt_mem el=xmemory.*
search_ckt_msx el=xmemory.xsubcircuit.*
search_ckt_analog el=xanalog.*
```

Using this example, the simulator combines the rules for `search_ckt_mem` and `search_ckt_msx` and applies them to the `xmemory.xsubcircuit` block, while applying `search_ckt_mem` to the remainder of the circuit in `xmemory`. In addition, the simulator applies the `search_ckt_analog` rule to the `xanalog` block and **search_ckt_msx** to the remaining portions of the circuit.

See Chapter 3, "ACE Configuration Commands" in the *PowerMill Reference Guide* for more information on the **search_ckt_mem** and **search_ckt_analog** configuration commands. For information on **search_ckt_logic**, see Chapter 2, "Configuration and Interactive Commands" in the *PowerMill Reference Guide*.

Controlling Node Sensitivity

Both the speed control and event sensitivity can be applied on a global basis (to the entire circuit, either all digital or all analog nodes) using the **set_sim_esv**, **set_sim_aesv**, **set_sim_spd**, and **set_sim_aspd** commands.

EXAMPLE 1:

```
set_sim_esv 0.01
```

This example sets the event resolution voltage of all digital nodes to be `0.01` V.

EXAMPLE 2:

```
set_sim_aesv 0.01
```

This example sets the event resolution voltage of all analog nodes to 0.01 V. Any voltage change, from its last event voltage, higher than 0.01 V will schedule an event, provided there is no local esv setting.

In some cases, different parts of the circuit could have different sensitivity requirements. In this case, you can apply the **set_node_spd** and **set_node_esv** commands to individual nodes.

EXAMPLE 3:

```
set_node_esv vco_in 0.01
```

This example sets the event resolution voltage at node `vco_in` to `0.01` V. Any voltage change, from its last event voltage, higher than `0.01` V schedules an event at node `vco_in`.

EXAMPLE 4:

```
set_node_spd VCO_IN 0.05
```

This command sets the spd value at node `VCO_IN` to `0.05` V.

Controlling the Simulation Time Resolution

Normally you will not need to modify the default simulation time resolution (10 ps). However, when simulating low frequency audio applications you might need to increase the time resolution (the minimum time step) to perhaps to 1 ns. If all nodes move slowly, a simulation time resolution of 10 ns or even 100 ns could provide sufficient accuracy with a high simulation speed. You can use the **set_sim_tres** command to modify the simulation time resolution.

EXAMPLE:

```
set_sim_tres 0.01ps
```

This example sets the minimum time resolution to `0.01` ps.

Once you choose the appropriate time resolution, you can apply additional time step control commands to control the time step.

Changing the Time Step Selection Parameter

You can use the **set_node_spd** and **set_node_aspd** (for analog) commands to change the size of the time step for a particular node.

EXAMPLE 1:

```
set_node_spd VCO_IN 0.05
```

This command sets the time step size parameter (spd) at node `VCO_IN` to `0.05` V. With this setting, the maximum voltage change each time step at node VCO_IN cannot exceed `0.05` V

Additionally, you can use the **set_sim_spd** and **set_sim_aspd** (for analog) to change the size of the time step selection parameter (spd) for the entire simulation.

EXAMPLE 2:

```
set_sim_spd 0.3
```

This example sets the simulation step size parameter (spd) to `0.3` V. With this setting, the maximum voltage change at each time step cannot exceed `0.3` V.

Applying Multiple Time Steps (Multi-Rate Simulations)

In mixed-speed systems, such as DSPs, or other discrete-sampled circuits with a high clock speed sampling low frequency signals, it might be necessary to use the multi-rate simulation capability. The ability to do multi-rate simulations means that you can apply different time steps to different parts of the circuit. Using multiple time steps helps you to obtain the maximum simulation speed from each circuit block. The simulation will speed up 2–3 times for some circuits, depending on the actual configuration of the circuit, when doing a multi-rate simulation. Use the **set_sim_subgroup** ACE configuration command to enable a multi-rate simulation. You can use this command only when running in SMS or synchronous mode.

Controlling Waveform Print Resolution

The PowerMill simulator allows you to change the time, voltage, or current resolution to the output file. For ACE, it is important to be able to alter the print resolution settings to get sufficient precision for post-processing functions such as .measure command processing or FFT processing.

You can use the following commands to change the output print resolution:

- **set_print_tres**
- **set_print_vres**
- **set_print_ires**

For complete reference information on these print resolution commands, see Chapter 2, "Configuration and Interactive Commands" in the *PowerMill Reference Guide*.

Applying Multiple Simulation Modes

You can apply different simulation modes on a local or block level using the following commands:

- **set_elem_acc**
  Use this command to apply a more accurate model and algorithm to specified MOSFETs.
- **set_node_pwl**
  Use this command to force the channel-connected stage containing a specified node to be simulated using the PWL algorithm.
- **set_elem_sms**
  Use this command to apply the Synchronous Matrix Solver (SMS) model to specified elements.
- **set_elem_sync**
  Use this command to apply the synchronous simulation algorithm to specified elements.

NOTE: Both the **set_elem_sync** command and the **set_elem_sms** command apply the synchronous mode. However, they use different models—the **set_elem_sync** command uses a simplified region-dependent gate capacitance model for MOS transistors at linear, saturation, and cut-off regions, while the **set_elem_sync** command uses a voltage-dependent gate capacitance model. In addition, the **set_elem_sms** command models the crosstalk Miller effect between gate and drain or source terminals.

Usually, the autodetection algorithm applies these commands as needed. However, there are situations when you might want to apply them manually (for example, when you need precise accuracy in digital circuits). You could use this technique, for example, when simulating an inverter ring oscillator.

Activating Double-Precision Mode

You can start the double-precision version of PowerMill for voltage and current using the -A command-line option. Under normal conditions, you won't need to use double precision. However, in some cases, double precision could be necessary. For example, highly damped PLLs with slow lock times, low static phase error, and low jitter requirements usually need to be simulated in the double-precision mode. Very high gain amplifiers or analog-to-digital circuits and digital-to-analog circuits beyond 12 bits might also require double precision.

Simulation Speed and Double-Precision

Using double-precision mode will slow down the simulation by approximately 10%, while increasing memory consumption by approximately 30%. Therefore, it is recommended that you do not use double-precision mode unless absolutely necessary.

Controlling Voltage and Current Resolution

There are no limits on the voltage and current resolution settings except those imposed by the floating-point (float) data type. The float data type is the default in single precision, which is an 8-bit mantissa and a 3-digit exponent. This means that the absolute lowest resolution available depends on the absolute value of a stored number.

For small voltages and currents there are more significant digits available at lower values. For example, if you have 1.00000000 V and you need to maintain accuracy to 1 nV, the nanovolt-level will be lost, that is,

1.000000000 V + 0.000000001 V = 1.000000000 V

However, if you have 1 mV instead, the 1 nV can be retained, that is,

0.001000000 V + 0.000000001 V = 0.001000001 V

Therefore, if you have large signals that require nanovolt accuracy and precision, you will need to use the double-precision data storage.

CAUTION: The higher precision printing has a direct impact on the size of the output file. If you try to print every node voltage in the circuit at the microvolt or nanovolt resolution level, you could easily fill up your disk.

Simulating BJTs

The ACE option extends PowerMill's capability to simulate BiCMOS circuits with bipolar junction transistors (BJTs). The BJT model is an implementation of the Gummel-Poon model used in SPICE. Both NPN and PNP transistors are supported, as well as diodes and diode-connected bipolar transistors.

Both vertical and lateral BJTs are supported. ACE is best suited for circuits containing sparse BJTs as drivers, or interface circuits. Simulation of circuits containing large blocks of BJTs will be slow, since those BJTs are simulated as a whole subcircuit. The simulation speed of small to medium-sized pure BJT circuits is in the same order as SPICE. If the circuit does not have large blocks of BJTs, the overall speed is faster than SPICE.

Procedure for Simulating BJTs

This section describes the process by which ACE simulates BJTs. This procedure includes several steps, some in preprocessing and some during run time.

- First, ACE reads the model from the netlist.

  It calculates some variables used during simulation and calculates the temperature effect, if the operating temperature is given.

- Next, ACE expands internal nodes and internal resistors if base, collector, and emitter resistors are specified.

  The base, collector, and emitter resistors are important in achieving convergence both at DC initialization and transient simulation since they limit the current through BJTs when the pn junction is forward biased. However, expanding those resistors and internal nodes significantly slows down the simulation.

- Next, ACE performs circuit partitioning on the BJTs as with an accurate MOS transistor, which includes its gate, drain, and source in the same stage (channel-connected subcircuit).

  The base of a BJT is included in the same stage with its collector and emitter. This partitioning scheme is necessary since base-emitter and base-collector junctions are strongly coupled. This effectively connects two channel-connected regions that would usually be separated when a normal MOS device is used. If a circuit contains only BJTs, all channel connected and base terminal-connected BJTs will form a large stage, which makes the simulation very inefficient.

The BJTs are simulated in the accurate element mode, which uses a different time step control and error control to assure simulation accuracy and stability.

Chapter 6

ACE Tutorials

Overview

The tutorials in this chapter are intended to exercise features and algorithms in the ACE simulation engine. The following tutorials are included in this chapter:

- Tutorial 1: CV Curve Generation
- Tutorial 2: Operational Amplifier Simulations
    - Voltage Follower Simulation
    - Step Input Simulation
    - Integrator Simulation
    - Differentiator Simulation
- Tutorial 3: PLL SPICE Macro Modeling
- Tutorial 4: Crystal Oscillator Simulation

These tutorials are written with the assumption that you know how to run vi (or another editor) and turboWave (or another waveform viewer). Therefore, the specific steps for opening and reopening files (loading signals) are not explicitly stated.

NOTE: The turboWave screens in this chapter were captured with altered colors and signal height. This was done to make the screens more readable in print format. The waveforms you will see in turboWave will look significantly different from the screens in this chapter.

Getting the Input Files

Before you can run any of the tutorials in this chapter, you need to locate and copy the required input files to your current working directory. The following procedure shows you how to do this.

1 Set the correct path to which the EPIC_HOME environment variable is set. If you need assistance, see your Systems Administrator.

2 Copy (recursively) the complete contents of *$EPIC_HOME/tutorials/ACE* to your local working directory.

```
cp -R $EPIC_HOME/tutorials/ACE .
```

This command copies the ACE directory into the current working directory (the period tells the **cp** command to copy to the current directory).

CAUTION: If there is an existing file or directory called *ACE* already in your current working directory, this command could overwrite that file or directory.

3 Verify that your newly copied ACE directory contains the following subdirectories: *crystal_osc*, *cv_curve*, *op_amps*, and *pll*. Each of these directories corresponds to and contains the files needed for a tutorial in this chapter.

SPICE Netlists

All tutorials in this chapter use SPICE format netlists. Conversion to EPIC format is not necessary.

Tutorial 1: CV Curve Generation

This tutorial demonstrates the procedure for checking the voltage dependent gate capacitance CV curve. Figure 1 provides a schematic of the CV curve test circuit.

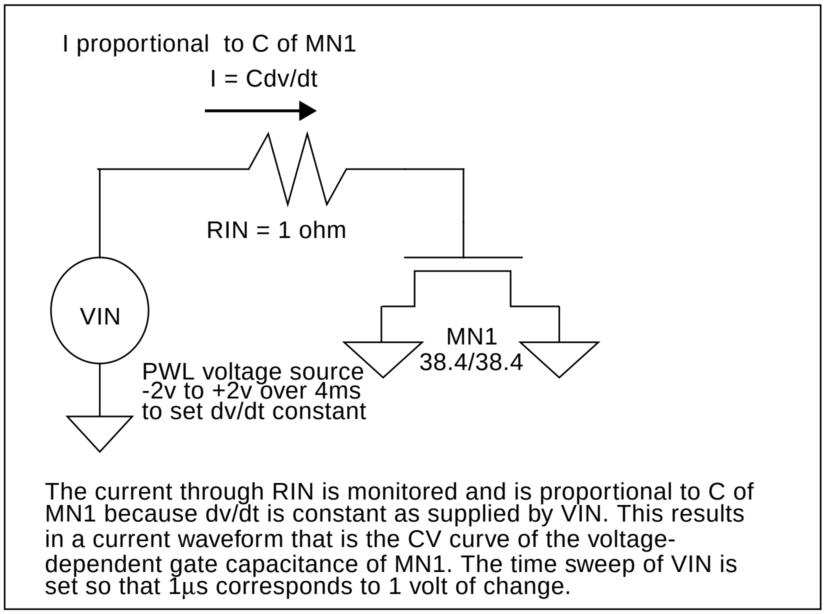

I proportional to C of MN1
I = Cdv/dt
RIN = 1 ohm
VIN
PWL voltage source
-2v to +2v over 4ms
to set dv/dt constant
MN1
38.4/38.4
The current through RIN is monitored and is proportional to C of MN1 because dv/dt is constant as supplied by VIN. This results in a current waveform that is the CV curve of the voltage-dependent gate capacitance of MN1. The time sweep of VIN is set so that 1µs corresponds to 1 volt of change.

Figure 1 Schematic of CV curve test circuit

Checking the voltage-dependent gate capacitance CV curve is done using the relation i=C*dv/dt. In the example circuit used in this tutorial, the current "i" is made proportional to the capacitance C by the input of a linear voltage ramp so that dv/dt is a constant. The voltage is swept from -2 V to +2 V to cover the depletion, accumulation, and inversion regions. Normally, the exact flat-band voltage point is in the negative voltage region below zero Vgs and is not critical to most simulations. A large area square transistor of 50 µm by 50 µm is used to avoid short channel effects and give a large enough capacitance to give a measurable current. A 1 ohm series resistor is used as the monitoring element. The current through this resistor is plotted to produce the CV curve.

Files Needed for this Tutorial

The following table lists the files needed for this tutorial. They are located in the *ACE/cv_curve* directory.

Filename	Description
cfg_cv	Batch run configuration file
CV_curve.spi	SPICE netlist for this tutorial
runcv	Run script for this tutorial
cmos35.mod	BSIM3 v3 model file

Procedure

Use the following procedure to run the CV curve tutorial.

1 Using vi (or another text editor) open and study the *CV_curve.spi* netlist file.

Notice that this is a very simple, short netlist. Notice that the PWL voltage source sweep is over 4 µs. This is done so that the time scale corresponds to the voltage scale. The full voltage sweep is 4 V from -2 V to +2 V. The 2 µs point will be the zero volt point of the sweep since the time starts at zero. You can subtract 2 from the time to get the voltage at any given point.

2 Run the simulation using the *runcv* script.

3 Using turboWave, load the file *CV_curve.out* file and then the i1(rin) current waveform and the v(in) voltage signal. You should see the curves shown in Figure 2.

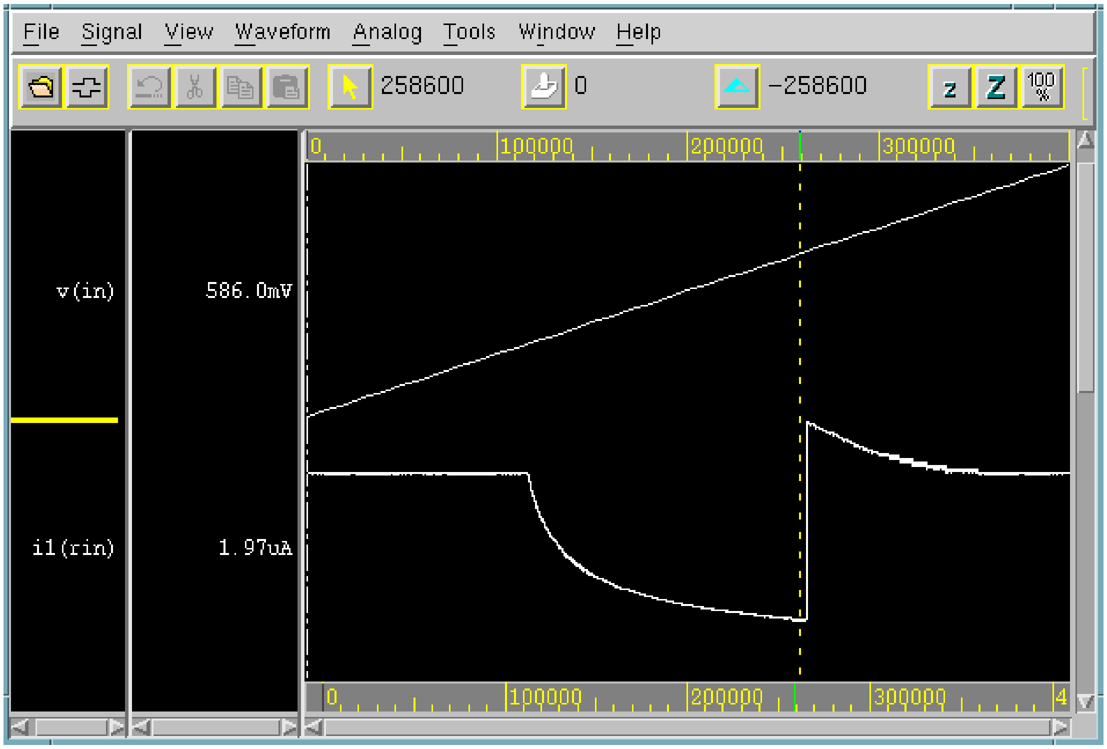


Figure 2 Waveforms of signals in CV_curve.out file

4 Calculate the capacitance at the minimum and maximum points using the i = C*dv/dt relation. Since you know the current (I) and dv/dt, you only need to calculate C.

Notice that the cursor position in Figure 2 shows 1.96 pF (minimum point).

Tutorial 2: Operational Amplifier Simulations

This tutorial utilizes a transistor-level op amp of very high gain in different configurations. The following configurations are used:

- Voltage follower
- Step input
- Integrator
- Differentiator

Figure 3 provides the op amp schematic used for this tutorial. This op amp has an open-loop gain of approximately 116 db.

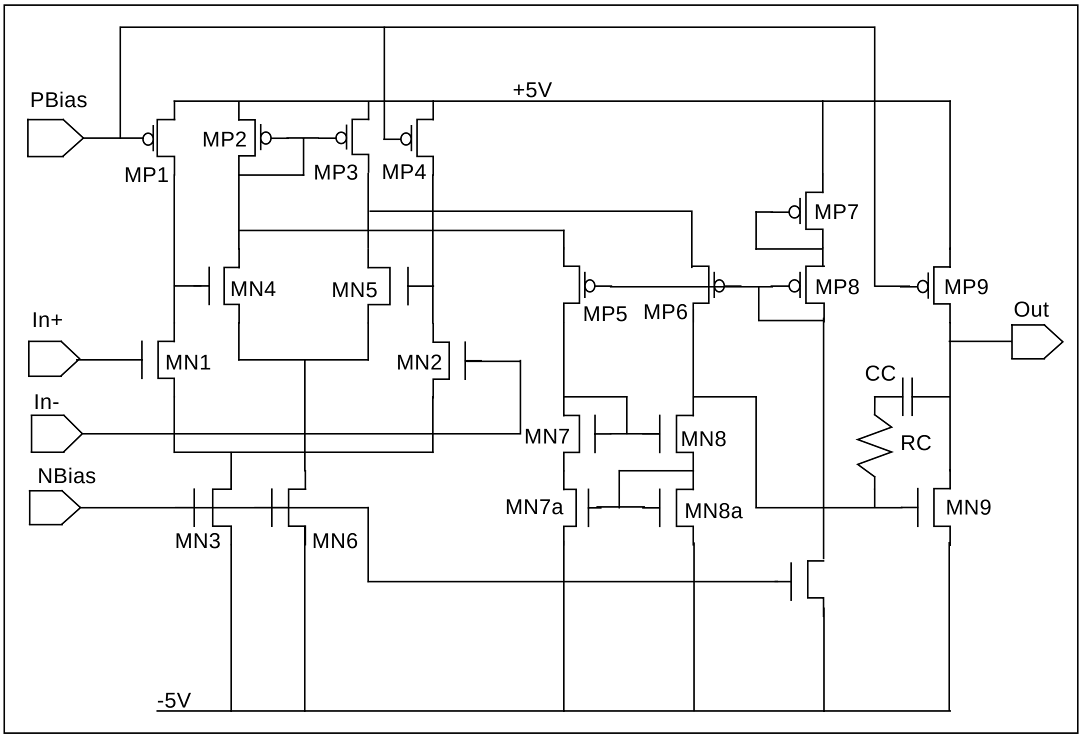


Figure 3 Schematic of op amp circuit

Voltage Follower Simulation

The following table lists the files needed for this simulation. They are located in the *ACE/op_amps* directory.

Filename	Description
cfg	Batch run configuration file
follower.spi	SPICE netlist for the voltage follower simulation
runfoll	Run script for the voltage follower simulation
cmos35.mod	BSIM3 v.3 SPICE model

Procedure

Use the following procedure to run the follower simulation.

1 Using vi, open the *follower.spi* netlist file.

Go to the bottom of the netlist and you will see the instantiation of the bias circuit block and the voltage follower connection of the op amp.

See Figure 4 for the voltage follower schematic.

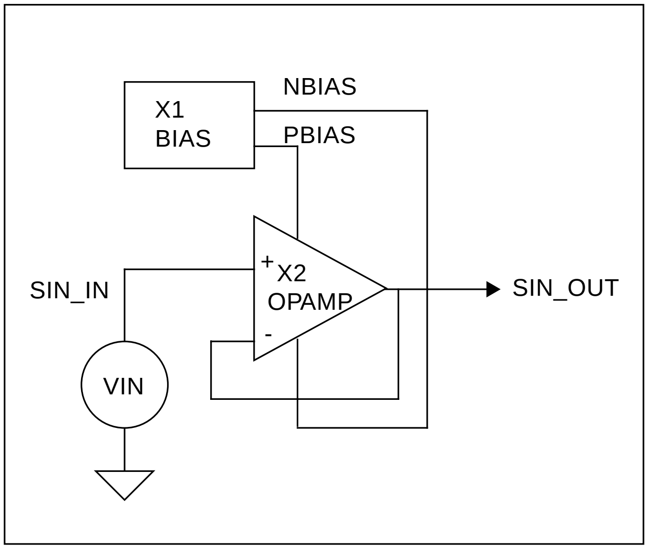


Figure 4 Schematic of voltage follower circuit

2 Run the simulation using the *runfoll* script.

3 Using turboWave, load the *follower.out* file.

4 Load the “sine_in” and “sine_out” signals (see Figure 5).

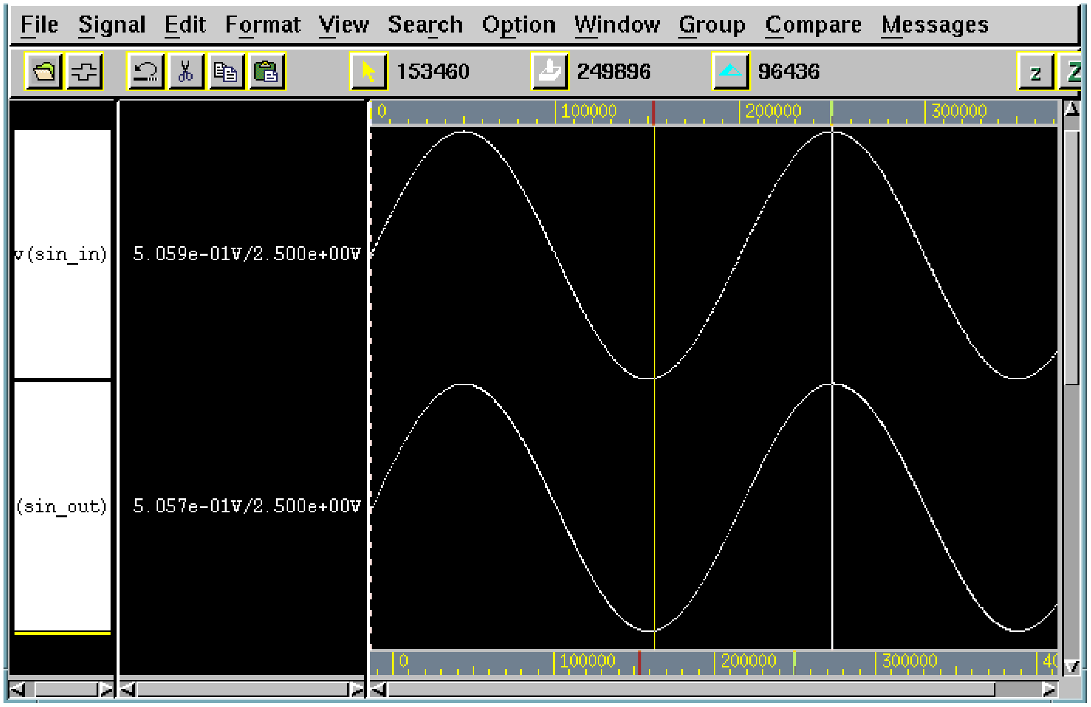


Figure 5 Waveforms of signals in follower.out file

Step Input Simulation

The following table lists the files needed for this simulation. They are located in the *ACE/op_amps* directory.

Filename	Description
cfg	Batch run configuration file
step.spi	SPICE netlist for the step input simulation
runstep	Run script for the step input simulation
typ_tech_nchan *typ_tech_pchan*	Technology files

Procedure

Use the following procedure to run the *step input* simulation.

1 Using vi, open the *step.spi* netlist file.

You will see that the connection is the same as the voltage follower. The difference is in the input waveform. In this case, the step response will be observed. The important parameters are the slew-rate going up and down, the settling time, any ringing effects, and the accuracy of the final value or gain error.

The schematic is shown in Figure 6.

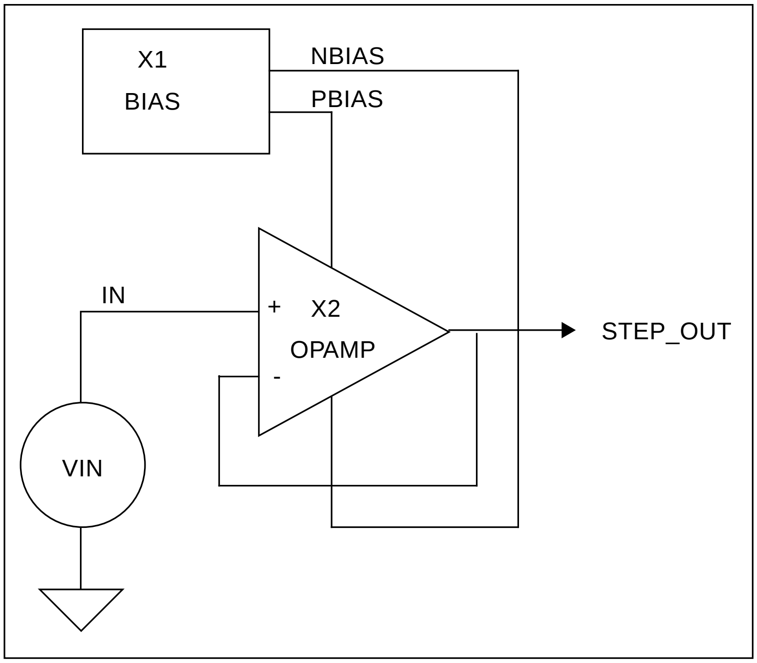


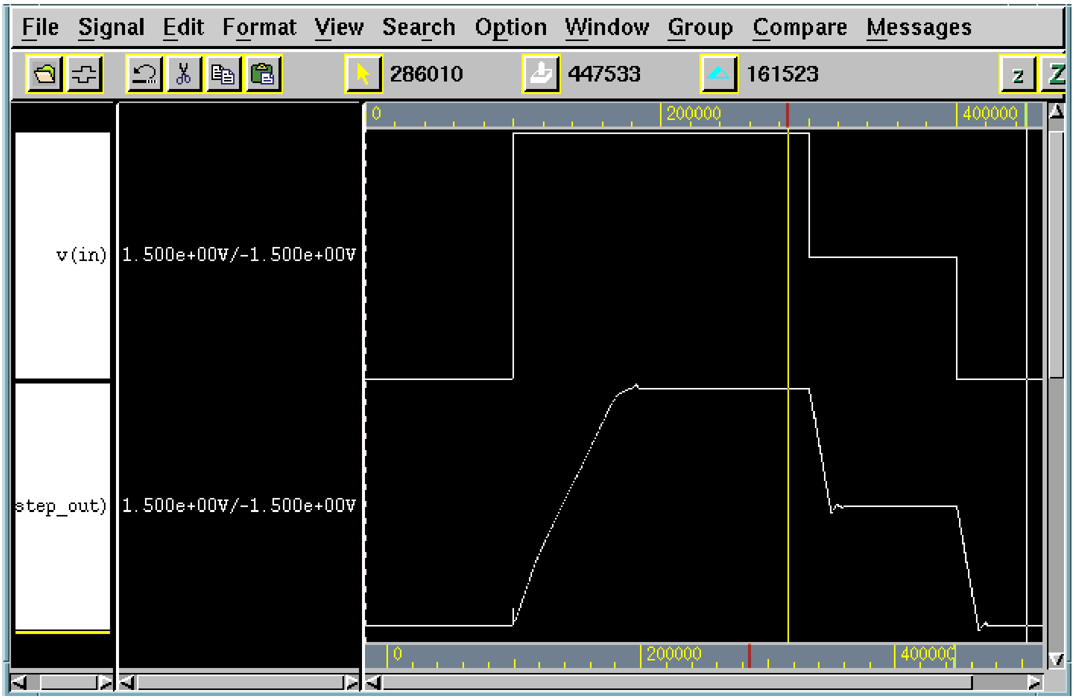
Figure 6 Schematic of step input circuit

2 Run the simulation using the *runstep* script.

3 Using turboWave, load the *step.out* file.

4 Load the “in” and “step_out” signals (see Figure 7).

There are three steps to the waveform [-1.5 to +1.5 to zero to -1.5 V].



```
v(in)        1.500e+00V/-1.500e+00V
step_out)    1.500e+00V/-1.500e+00V
```

Figure 7 Waveforms of signals in the step input simulation

Integrator Simulation

The following table lists the files needed for this simulation. They are located in the *ACE/op_amps* directory.

Filename	Description
cfg	Batch run configuration file
integr.spi	SPICE netlist for the integrator simulation
runint	Run script for the integrator simulation
typ_tech_nchan *typ_tech_pchan*	Technology files

Procedure

Use the following procedure to run the *integrator* simulation.

1 Using vi, load the *integr.spi* netlist file.

Figure 8 provides the schematic of this circuit.

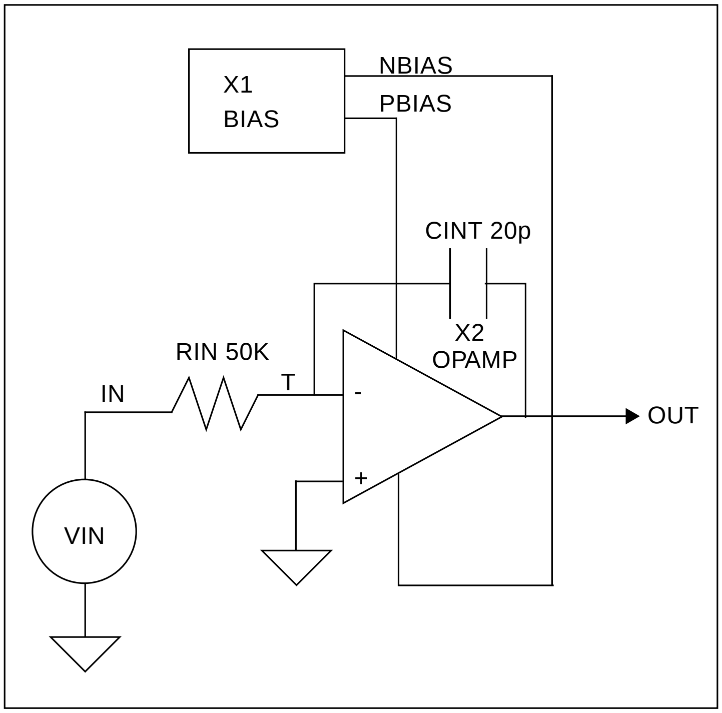


Figure 8 Schematic of Integrator circuit

This circuit is a simple integrator using a 50K resistor and a 20 pF capacitor. The input waveform is a 250Khz square wave between -2 V and +2 V.

2 Run the simulation using the *runint* script.

3 Using turboWave, load the *integr.out* file.

4 Load the “in” and “out” signals (see Figure 9).

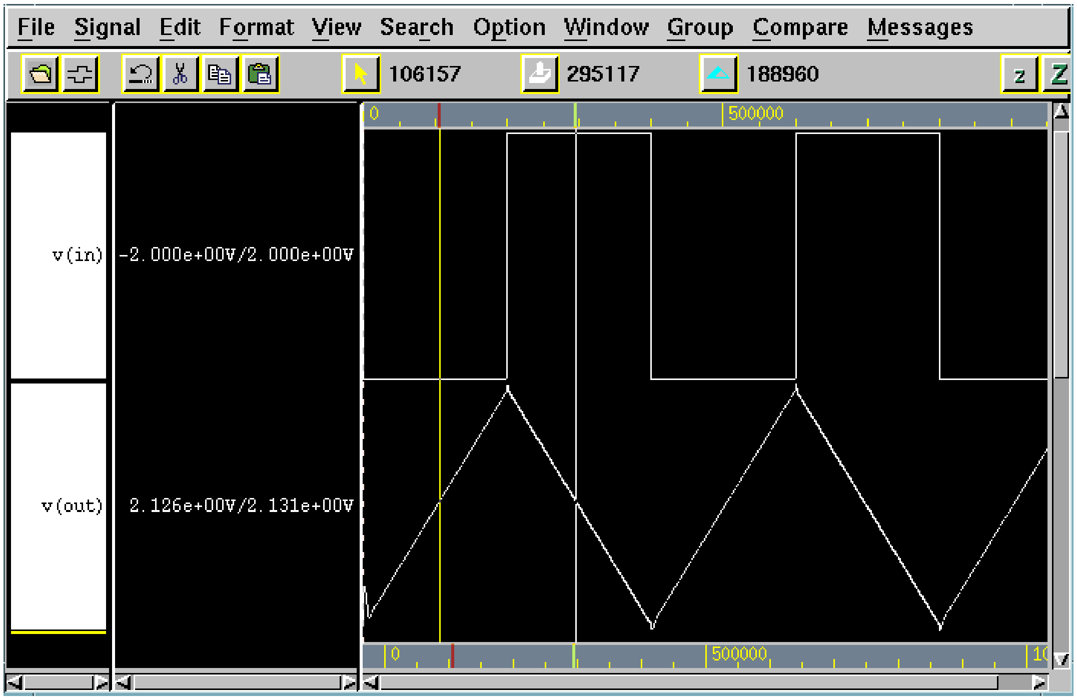


Figure 9 Waveforms of signals in integr.out file

As expected, the “out” signal is a triangle wave. There is some initialization error at the beginning of the simulation where the waveform is not tracking as it should. As you can see, this clears out after short time.

Differentiator Simulation

The following table lists the files needed for this simulation. They are located in the *ACE/op_amps* directory.

Filename	Description
cfg	Batch run configuration file
diff.spi	SPICE netlist for the differentiator simulation

Filename	Description
rundiff	Run script for the differentiator simulation
typ_tech_nchan *typ_tech_pchan*	Technology files

Procedure

Use the following procedure to run the *differentiator* simulation.

1 Using vi, open the *diff.spi* netlist file.

Figure 10 provides the schematic for this circuit.

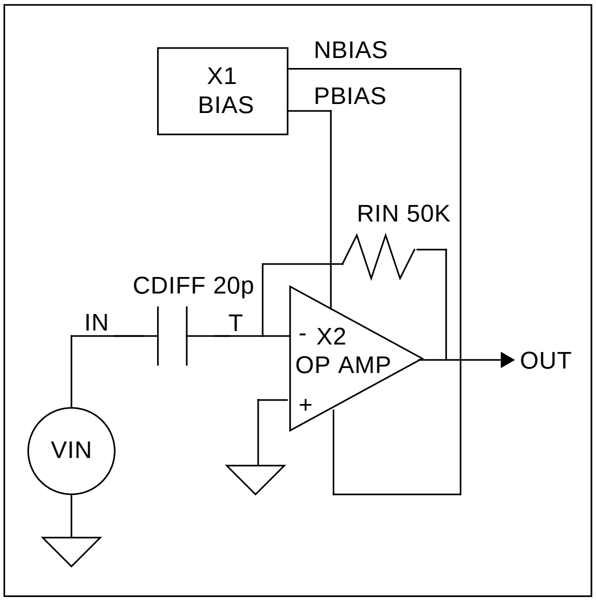


Figure 10 Schematic of differentiator circuit

This circuit is the opposite of the integrator. As in the integrator, this circuit uses a 50K ohm resistor and 20 pF capacitor. The compensation for the op amp was changed to allow this circuit to function; a differentiator configuration can oscillate more easily if the op amp is not properly compensated for the task. Check the compensation capacitor in the subcircuit definition.

2 Run the simulation using the *rundiff* script.

3 Using turboWave, load the *diff.out* file.

4 Load the “in” and “out” signals (see Figure 11).

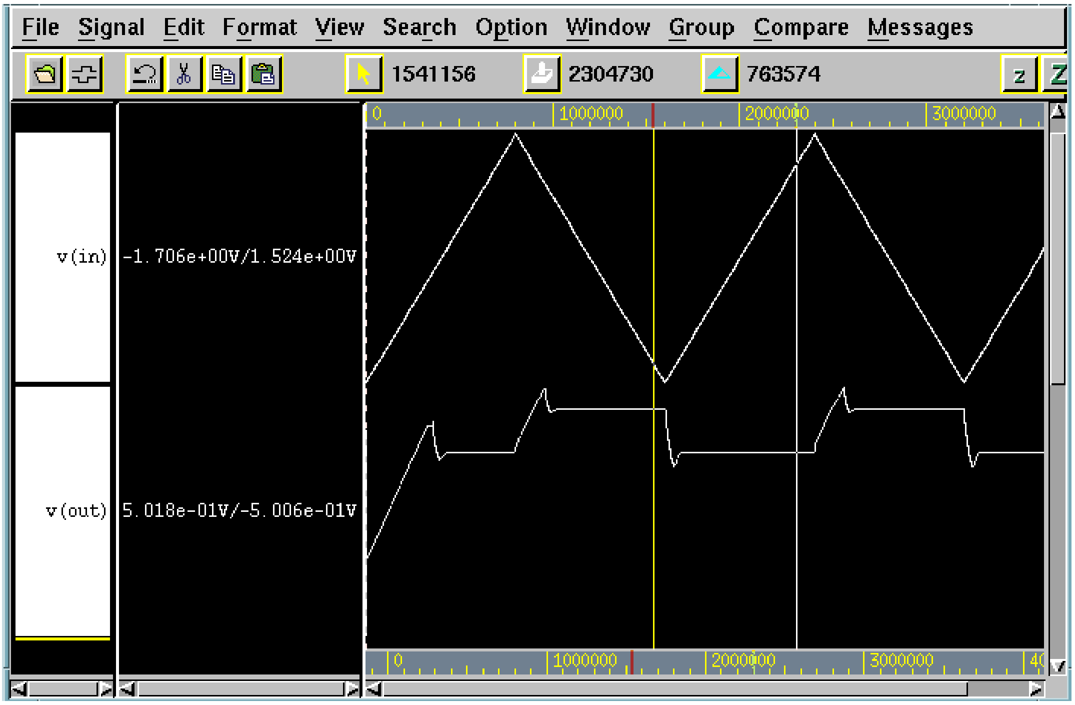


Figure 11 Waveforms of signals in diff.out file

You will notice slew-rate limiting effects and settling time limit effects. The slope of the input triangle waveform is 0.5 V per microsecond and should give plus and minus 0.5 V on the output after it settles.

Tutorial 3: PLL SPICE Macro Modeling

This tutorial uses a PLL SPICE macro model. It models frequency as voltage. That is, voltage is an analog of frequency.

In this tutorial, you will learn how to model the system behavior of a PLL by using controlled sources to model the different sections of the system. The phase detector and charge pump are modeled with a voltage controlled current source. The 1/s integrator term is modeled with a voltage controlled current source with a gain of 1 driving a 1-farad cap. The output of this cap drives a voltage-controlled voltage source. This voltage source has all the gain terms of the loop lumped together, except for the charge-pump current and the loop filter. The loop filter components are used as they are in the RC network.

Files Needed for this Tutorial

The following table lists the files needed for this tutorial. They are located in the *ACE/pll* directory.

Filename	Description
cfg	Batch run configuration file.
pll2gm.spi	SPICE netlist for the PLL macro model simulation.
runpllm	Run script for the PLL macro model simulation.

Procedure

Use the following procedure to run the PLL SPICE macro modeling tutorial.

1 Study the schematic in Figure 12.

This schematic shows the controlled sources as triangle symbols, similar to op amp symbols. There are two voltage-controlled current sources and two voltage-controlled voltage sources. The loop filter components are entered as they would normally appear in the transistor-level netlist.

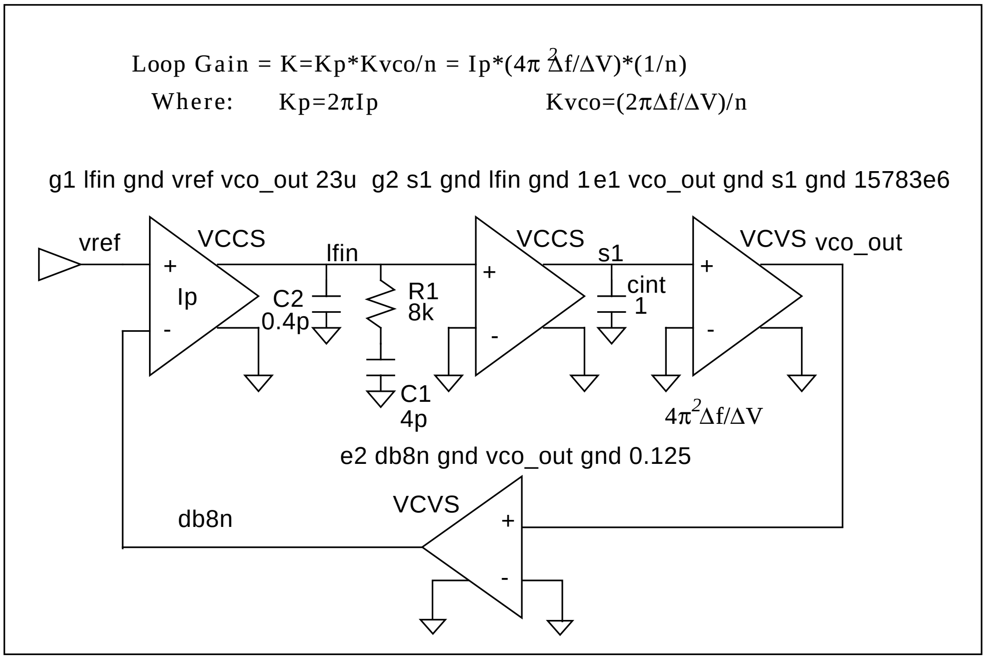

Loop Gain = K=Kp*Kvco/n = Ip*(4π²Δf/ΔV)*(1/n)
Where: Kp=2πIp Kvco=(2πΔf/ΔV)/n
g1 lfin gnd vref vco_out 23u
g2 s1 gnd lfin gnd 1
e1 vco_out gnd s1 gnd 15783e6
vref
VCCS
+
Ip
-
lfin
C2
0.4p
R1
8k
C1
4p
VCCS
+
-
s1
cint
1
VCVS
+
-
vco_out
4π²Δf/ΔV
e2 db8n gnd vco_out gnd 0.125
VCVS
+
-
db8n

Figure 12 Schematic of PLL SPICE macro model

This schematic includes the loop gain equation and a representation of each piece of the loop gain parameters. Since the total loop gain is the product of each contributing segment, you can separate terms and place them anywhere in the loop.

To make the circuit easy to modify, the charge pump current is placed by itself as the gain for the phase detector model in the first VCCS element.

The feedback divide ratio is placed in the second VCVS element as 1/n. In this case, it is 1/8 or 0.125. The balance of the loop gain is put into the first VCVS that models the VCO.

2 Using vi, look at *pll2gm.spi* netlist file. Notice that the r1 resistor has a value of 1k.

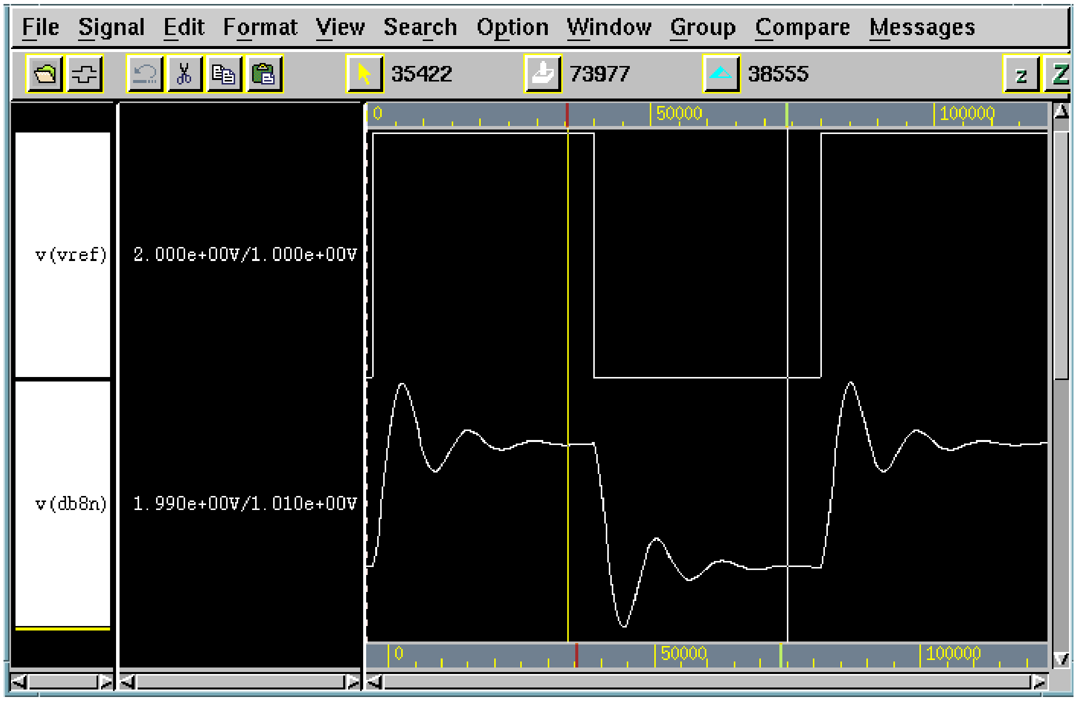
This is the incorrect value used for the first run in this tutorial. The result is an underdamped system response to a step input.

3 Run the simulation using the *runpllm* script.

4 Using turboWave, load the *pll2gm.out* file.

5 Display the “vref” and “db8n”signals. The “vref” signal is the input voltage and the “db8n” signal is the feedback. You will see that “db8n” shows underdamped or ringing behavior.

See Figure 13 for the expected waveforms.



v(vref) 2.000e+00V/1.000e+00V

v(db8n) 1.990e+00V/1.010e+00V

Figure 13 Waveforms of signals in pll2gm.out file (underdamped)

6 Using vi to edit the *pll2gm.spi* file, comment out the `r1` line with the `1k` resistor.

7 Uncomment the other `r1` line containing the `8k` resistor.

With this change, the damping factor is now near critical damping.

8 Run the simulation again using the same *runpllm* script.

9 In turboWave, select **File→Reload**.

Figure 14 shows the new waveforms.

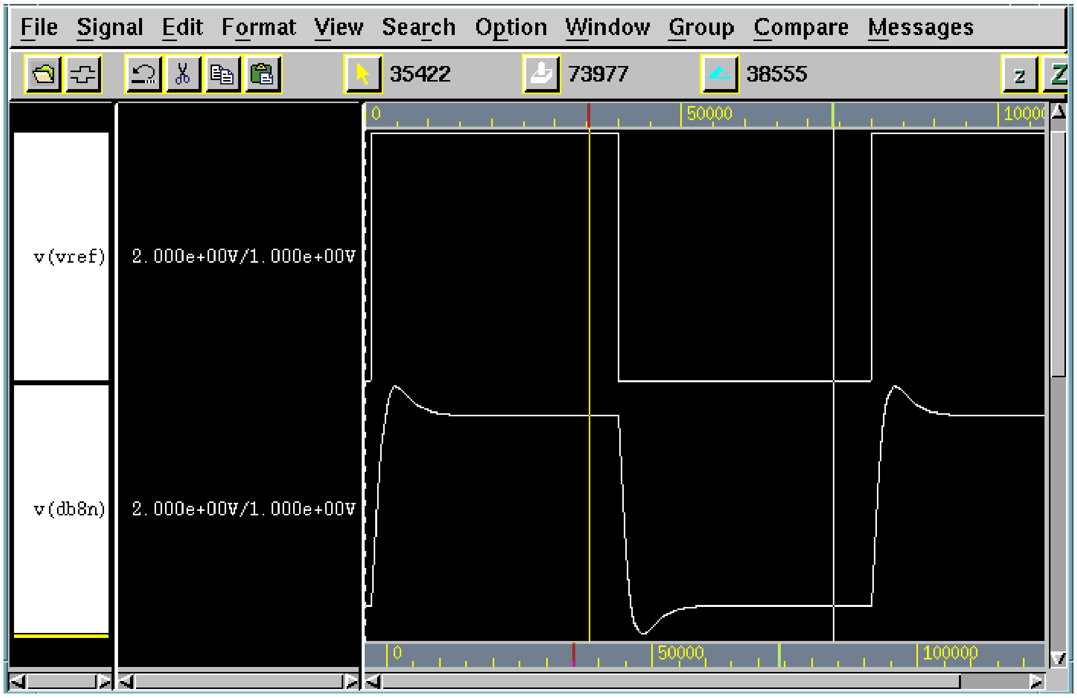


Figure 14 Waveforms of signals in pll2gm.out file (critically damped)

This will reload the previously selected signals. The “db8n” signal now shows a well-behaved smoothly settling waveform and the ringing behavior is gone.

10 Continue entering different values for r1 and C1 and rerun the simulation.

You will see how quickly a macro model of a PLL can give indications of system response. This provides a fast easy way to do a sanity check on PLL parameters such as loop filter values, charge-pump current, and the effects of different feedback divider ratios.

Tutorial 4: Crystal Oscillator Simulation

This tutorial demonstrates the effect of the **set_sim_trap** configuration command to apply the trapezoidal integration. This command is intended for use with LC circuits and inductors where free oscillations should not be suppressed or damped. This is critical for crystal oscillator models that use an RLC model. The **set_sim_trap** command allows these to start up normally. If the default Euler integration method is used, the crystal oscillator will not start because the free oscillations are damped out. The **set_sim_trap** command will not work for very high Q crystals such as a 32 KHz crystal model. This is due to the very high equivalent inductance, very small equivalent capacitance, and high equivalent series resistance.

Files Needed for this Tutorial

The following table lists the files needed for this tutorial. They are located in the *crystal_osc* directory. (You were instructed, at the beginning of this chapter, to copy this directory from *$EPIC_HOME/tutorials/ACE*.)

Filename	Description
cfg1, cfg2, cfg3	Batch run configuration files
crystal_osc.spi	SPICE netlist for the crystal oscillation simulation
run1, run2, run3	Run scripts for the crystal oscillator simulation
cmos35.mod	BSIM3 v.3 model file

Procedure

Use the following procedure to run the crystal oscillator tutorial.

1 Run the simulation using the *run1* script.

This case uses the default mode and will run very fast because the oscillator will not start. You will see the normal biasing curve as the inverter biases to the operating point.

2. View the contents of the *cfg2* file and notice the addition of the `set_sim_trap` command.
3. Run the next simulation using the *run2* script.

   You will see the oscillator start up at about the 200 µs point. The oscillations grow exponentially until they are about full amplitude around the simulation end time of 400 µs. This simulation does not run particularly fast due to the use of the default time resolution (tres) of 10 ps.
4. View the contents of the *cfg3* file and notice the addition of the `set_sim_tres 1n` command. This relaxes the minimum event time resolution to 1ns from 10 ps.
5. Run the simulation using the *run3* script.

   Notice that this simulation runs much faster than that run with the *run2* script.
6. Using turboWave, load the following files: *xtal1.out*, *xtal2.out*, and *xtal3.out*.
7. Load the signals you want to see (see Figure 15).

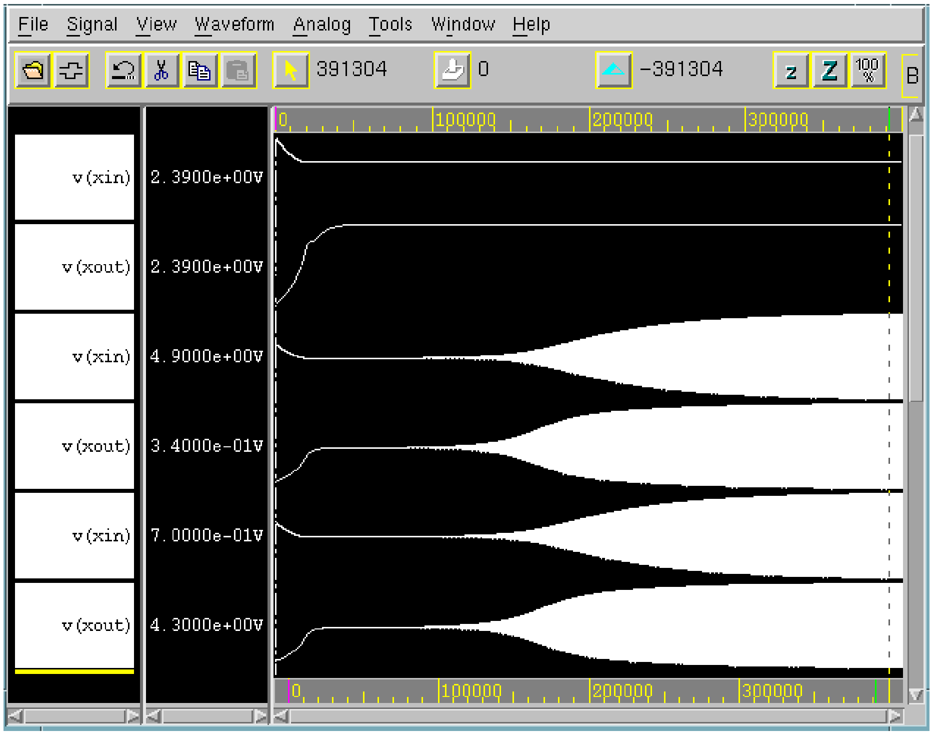


Figure 15 Waveforms of signals from .out files

Discussion

Normally, it is not necessary to simulate crystal oscillators because their behavior is well known. The small signal transconductance of the inverter required for start-up can be easily calculated and the MOS devices sized accordingly. You can effectively simulate crystal models in the megahertz range using the **set_sim_trap** configuration command. It is recommended that you not simulate crystal oscillators in the full-chip configuration. It is better to drive the input side of the inverter with a SPICE-style sinewave source. This produces realistic behavior of the oscillator input clock buffers to the interior of the circuit and accurate power consumption numbers. Leave the external load capacitors in place and remove the crystal model to prevent the simulator from trying to handle the RLC model and slow the simulation down unnecessarily.

Chapter 7

Using the PowerMill Graphical Analyst

Overview

The PowerMill simulator's Graphical Analyst (GA) provides an integrated approach to simulating circuits and analyzing results using the PowerMill simulator.

We value your comments and encourage you to give us feedback. You can give us direct feedback by selecting **Feedback** from the **File** menu of the **PowerMill GA** main window. See "File Menu" on page 165 for details.

This chapter explains how you can use the PowerMill Graphical Analyst to do the following tasks:

- Running a Basic PowerMill Simulation
- Starting the Graphical Analyst
- Setting Up a Run
- Running a Simulation
- Analyzing a Circuit

The following tools are available for circuit analysis:

- Power Consumption
- Power Histogram
- DC Path Browser
- Power Diagnostics Browser
- Hierarchy Browser

Running a Basic PowerMill Simulation

This section describes the procedure for running a basic PowerMill simulation using the Graphical Analyst.

1. Set up your environment so that you can access the directory containing the current PowerMill release (see Chapter 2, "Getting Started" for details).
2. Have the following input files in your current directory:

- A netlist file in SPICE, HSPICE (xxx.*spi*) or EPIC format (xxx.*ntl*).
- One or more technology files (xxx.*tech*).
- A vector file (optional).
- Any additional configuration files (optional).

3. Start the Graphical Analyst by typing one of the following commands at the UNIX prompt:

```
pwga
powrmill -ga
```

4. Select **Simulation Setup→Design Data Setup**.
5. Select the applicable netlist, technology, and configuration files.
6. Click **OK**.

   This will create a *.wrk* file by default. After it is created, you can specify it when invoking the Graphical Analyst and the **Setup Information** will be automatically entered by PowerMill.

7. Click the **Start Simulation** button. This invokes the simulation in the *powrmill_pwga* directory. This directory contains the configuration files automatically created by the GUI as well as the following output files: *powrmill.log*, *powrmill.err*, and *powrmill.out*
8. Click the **Result Analysis** button to access the analysis tools.
9. When you are finished viewing the results, close the window.

Starting the Graphical Analyst

As stated previously, you can start the PowerMill Graphical Analyst (GA) by typing one of the following commands at the UNIX prompt:

```
pwga
powrmill -ga
```

If you have previously simulated a circuit using the GA, the next time you start pwga, it will, by default, load the last-run work file and display the results of the most recent simulation.

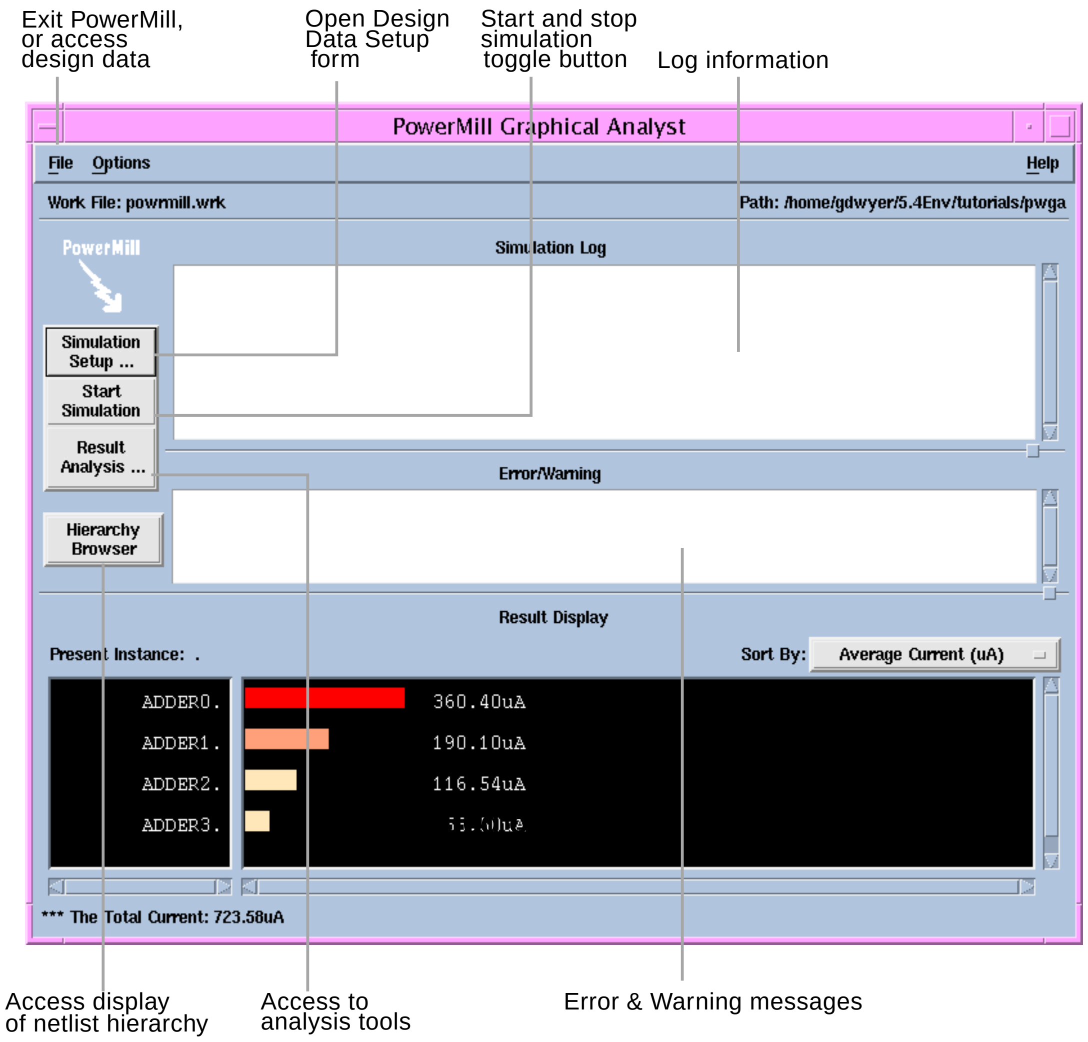
EXAMPLE:

```
pwga run_script.wrk
```

PowerMill uses the work file to determine the location of the PowerMill results.

Main Window

After you invoke the PowerMill Graphical Analyst, the **PowerMill GA** main window appears. This window is used to set up, start, and stop a simulation, and access analysis tools. It also displays log, warning, and power consumption information (see Figure 1).



Work File: powrmill.wrk

Path: /home/gdwyer/5.4Env/tutorials/pwga

Present Instance: .

```
ADDER0.    360.40uA
ADDER1.    190.10uA
ADDER2.    116.54uA
ADDER3.     55.[illegible]uA
```

*** The Total Current: 723.58uA

Figure 1 PowerMill GA main window

File Menu

You can use the **File** menu to start setting up a simulation run, send feedback via e-mail to Synopsys, save the window settings, or to exit PowerMill.

- To start setting up a run, select **File→Design Data Setup**. See the "Setting Up a Run" on page 167 for more information on this process.
- To send feedback to Synopsys, select **File→Send Feedback**. This pops open an e-mail window that allows you to enter your comments.
- To save the size and placement of the PowerMill GA windows, you can select **File→Save Window**.
- To exit the PowerMill Graphical Analyst, select **File→Exit**.

Options Menu

The **Options** menu lets you either choose the measurement unit you want to use for your analysis, or select your preferred waveform viewer.

Select **Options**→ **u (micro)** to view data in micro units, or select **m (milli)** to view data in milli units in the Results Display area of the **PowerMill GA** main window as well as the Power Histogram Display.

Select **turboWave** to use the turboWave viewer, or **SimWave** to use the SimWave viewer.

Help Menu

The **Help** menu gives you access to help information on the PowerMill Graphical Analyst.

Simulation Setup Button

The **Simulation Setup** button, located on the middle left side of the **PowerMill GA** main window, is one of two ways you can begin setting up for a simulation run. You can also select **Design Data Setup** from the **File** menu. Both of these actions cause the **Design Data Setup** form to appear. See “Setting Up a Run” on page 167 for more information on this process.

Start Simulation Button

The **Start Simulation** button is used to invoke a simulation run. See “Running a Simulation” on page 179 for more information.

Result Analysis Button

The **Result Analysis** button is used to access the PowerMill GA analysis tools after you have completed a simulation run. See “Result Analysis Button” on page 166 for more information.

Hierarchy Browser Button

The **Hierarchy Browser** button gives you access to a visual display of the netlist hierarchy. See “Hierarchy Browser” on page 193 for more information.

Simulation Log Display

The Simulation Log text field, located near the top of the main window, displays log information as the simulation progresses. It will notify you when a simulation is finished.

Error/Warning Display

The **Error/Warning** display shows you error and warning messages as a simulation progresses.

Result Display

The **Result Display**, located near the bottom of the main window, gives you visual power consumption information. See “Analyzing a Circuit” on page 180 for more information.

Sort By Menu

The **Sort By** menu, located above the **Result Display**, allows you to sort the kind of power consumption you want to display.

Setting Up a Run

Before running a simulation, you must specify the netlist and technology files PowerMill will use. You'll also need to specify power supply information for the run, block set up variables, and the power diagnosis checks you want to perform.

1 To start the set up process, click the **Simulation Setup** button located at the middle left side of the **PowerMill GA** main window.

The **Simulation Setup** form will appear.

Figure 2 is an example of the **Simulation Setup** form.

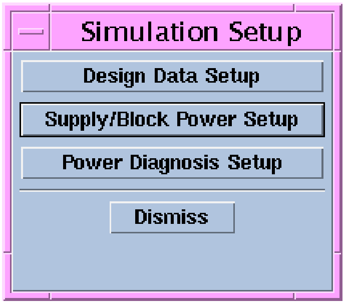
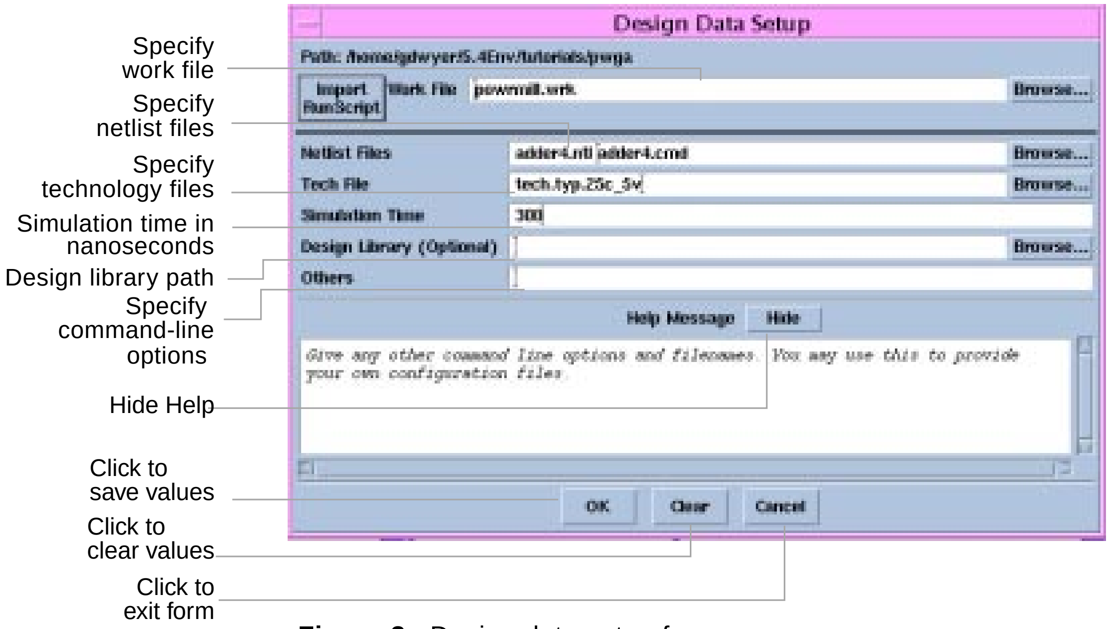


Figure 2 Simulation setup form

2 Click either **Design Data Setup** in the **Simulation Setup** form, or select **Design Data Setup** from the **File** menu.

The **Design Data Setup** form, as shown in Figure 3, will appear.



Path: /home/gdwyer/5.4Env/tutorials/pwga

Work File	powrmill.wrk
Netlist Files	adder4.ntl adder4.cmd
Tech File	tech.typ.25c_5v
Simulation Time	300
Design Library (Optional)	
Others	

Give any other command line options and filenames. You may use this to provide your own configuration files.

Figure 3 Design data setup form

You can get to this form from either the **File** menu or the **Simulation Setup** form.

Specifying a Work File

A work file is used to store information associated with a particular simulation run. After a run is completed, the data can be read directly from the work file without having to enter information again.

Work file names always end with a *.wrk* extension. After each simulation, a results directory with the same name as the work file (with a *_pwga* extension) is created. All simulation output files, as well as files generated by the Graphical Analyst are placed in the results directory.

By default, the PowerMill GA assumes a work file called *powrmill.wrk*. The results of the simulation are placed in the *powrmill_pwga* directory.

To store the results of multiple runs, you can change the *.wrk* file name. This will create a directory with a corresponding name. The directory contains the results of a particular simulation.

All paths specified in the **Design Data Setup** form are assumed to be relative to the directory containing the work file.

You can create a work file, or select an existing work file in the **Design Data Setup** form using one of the following methods:

- Type a new work file name in the **Work File** text field.
- Use an existing work file name if it already appears in the **Work File** text field.
- Click the **Browser...** button located to the right of the **Work File** text field.

This pops up the **File Finder** form. You can use this form to select an existing work file. Figure 4 is a snapshot of the **File Finder** form.

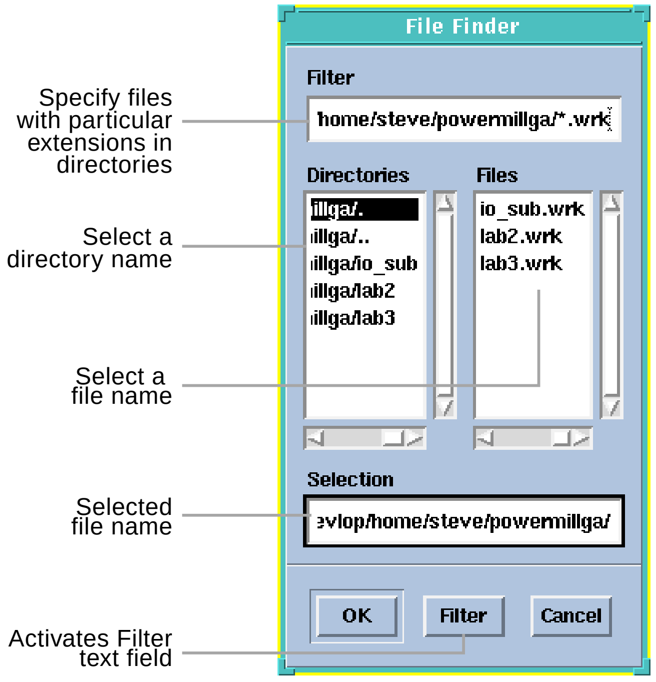


Figure 4 File finder form

Using the File Finder Form

The **File Finder** form gives you easy access to the files you need to run your simulation.

The **Filter** text field, located at the top of the form, lets you designate files with specific extensions.

1 Type `*.wrk` at the end of the path.

2 Click the **Filter** button, located at the bottom of the form, to see all files in the directory that have a *.wrk* extension.

The **Directories** column is used to select a directory containing the file you want to use. You can change the directory being viewed by double-clicking on a directory name.

A list of files in the directory you selected appears in the **Files** column.

The **Selection** field at the bottom of the **File Finder** form shows you the full path and name of the file you selected.

3 Click the **OK** button to save your selection, and close the **File Finder** form.

Importing Existing Run Scripts

The Graphical Analyst accepts existing PowerMill run scripts.

1 Click the **Import RunScript** button.

The **File Finder** dialog box will appear.

2 Type in the name of an existing run script in the **Selection** dialog box. The options specified in the run script are automatically entered into the appropriate fields, and a work file is created.

3 Click **OK** button to save your settings and close the form.

Specifying Netlist and Technology Files

You can either select the netlist and technology files manually or use the **File Finder** form.

1 To reopen the **File Finder** form, click the appropriate **Browse...** button located next to the **Netlist Files** or **Tech File** text fields in the **Design Data Setup** form.

2 Click the **OK** button to save your settings and close the form.

NOTE: To automatically generate technology files, you can specify the -z or -r options in the **Others** field of the **Design Data Setup** form.

Setting the Simulation Time

Enter the number of nanoseconds you want the simulation to run for in the **Simulation Time** text field of the **Design Data Setup** form. The default is 1000 ns.

Design Library

Specify the path to your design library in the **Design Library** text field, this is optional.

Command-Line Options

The **Others** text field of the **Design Data Setup** form is used to specify powrmill command line options.

Saving, Clearing or Cancelling Your Changes

1. After you've filled in the appropriate fields of the **Design Data Setup** form, click the **OK** button.

   This saves your settings, compiles the design, and closes the form. It also created a batch run script with a *.batch* extension.

   The batch run script can be run from the UNIX command line. It automatically places the results in the appropriate directory so that you can view the results of a batch simulation at any time.

2. Click the **Clear** button if you want to clear all information on the form.

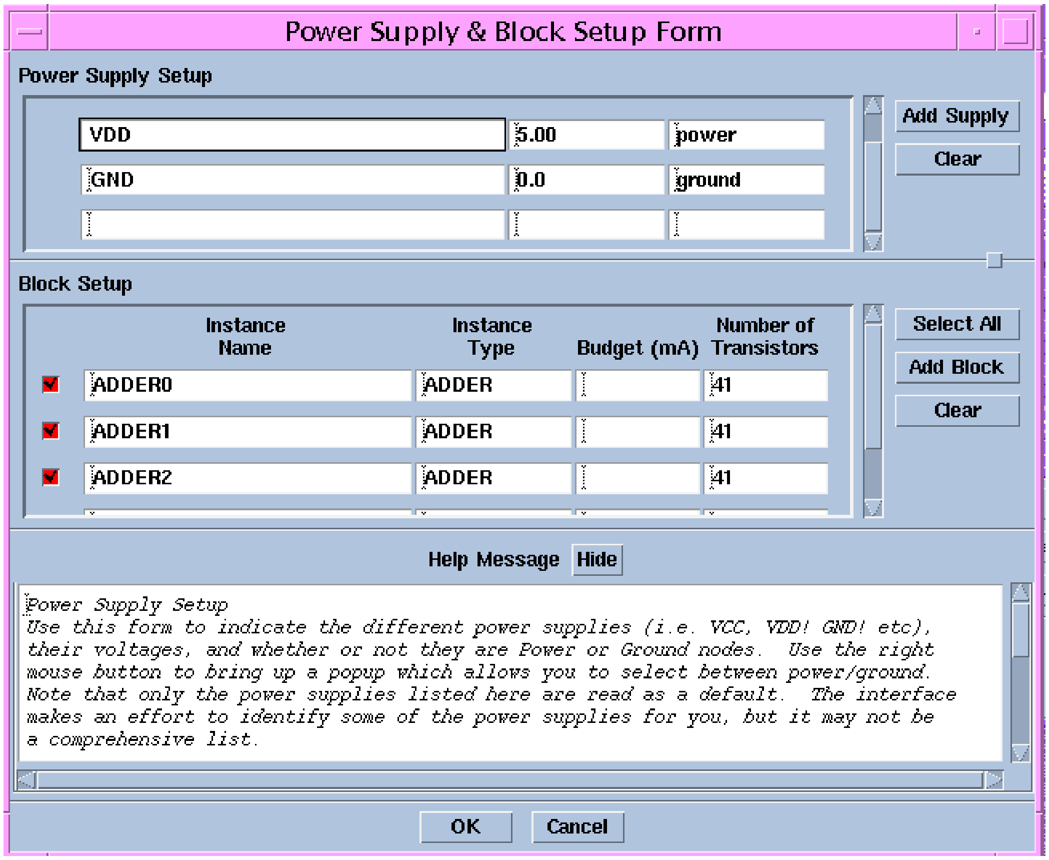
3. Click the **Cancel** button if you want to close the form without saving your changes.

Power Supply Setup

Use the **Power Supply & Block Setup** form to assign alternate voltages to the supply nets.

1 In the **Simulation Setup** form, click the **Power Supply & Block Setup** button.

Figure 5 shows the **Power Supply & Block Setup** form.



Power Supply Setup

VDD	5.00	power
GND	0.0	ground

Block Setup

	Instance Name	Instance Type	Budget (mA)	Number of Transistors
✓	ADDER0	ADDER		41
✓	ADDER1	ADDER		41
✓	ADDER2	ADDER		41

Power Supply Setup
Use this form to indicate the different power supplies (i.e. VCC, VDD! GND! etc), their voltages, and whether or not they are Power or Ground nodes. Use the right mouse button to bring up a popup which allows you to select between power/ground. Note that only the power supplies listed here are read as a default. The interface makes an effort to identify some of the power supplies for you, but it may not be a comprehensive list.

Figure 5 Power supply & block setup form

The top section of the **Power Supply & Block Setup** form identifies the supply nets and voltages for each block of the circuit you want to simulate. The settings that appear in the

Voltage column are specified in the technology file, or from a previously-saved modification. You can modify the voltage for any net. The GND is automatically set to 0V.

The **Power/Ground** column allows you to specify whether the supply net is a power net, or a ground net. By default, supply nets greater than 0 V are assumed to be power nets. Nets that are less than or equal to 0 V are assumed to be ground nodes.

1 To specify whether a net is a ground or power node, click the right mouse button in the power/ground field.

A popup dialog box appears.

2 Select either power or ground.

3 Click the **Add Supply** button to add supply nets.

The **Clear** button clears all the voltage values and any new power supply nets you added. The supply nets derived from the netlist remain in the form, but are no longer assigned voltage values.

Block Setup

You can also use the **Power Supply & Block Setup** to specify the blocks you want to monitor, and set up power budgeting checks.

The text fields in the bottom section of the form list the top-level blocks in the circuit and related information. If there are more than 100 blocks at the top level, no blocks will be displayed. In this case, you must manually select the blocks you want to include in the power calculation.

Transistors are not displayed in the form. If the design is flat, (that is, no hierarchy), then no blocks are selected. A popup warning will inform you that there are no top-level blocks.

Selecting Blocks

1 To select blocks, click the toggle button located to the left of the **Block Name** text fields.

A block is active when the button is red.

2 Click the **Set All** button at the bottom of the form to activate all blocks.

By default, all blocks at the top level of the design are selected.

3 Specify the budget violation threshold for each block in the **Budget** text fields.

The **Add Block** button at the bottom of the form adds a set of text fields for a new block. You can type in lower-level block names or drag and drop names from the **Hierarchy Browser**.

Saving, Clearing or Cancelling Your Changes

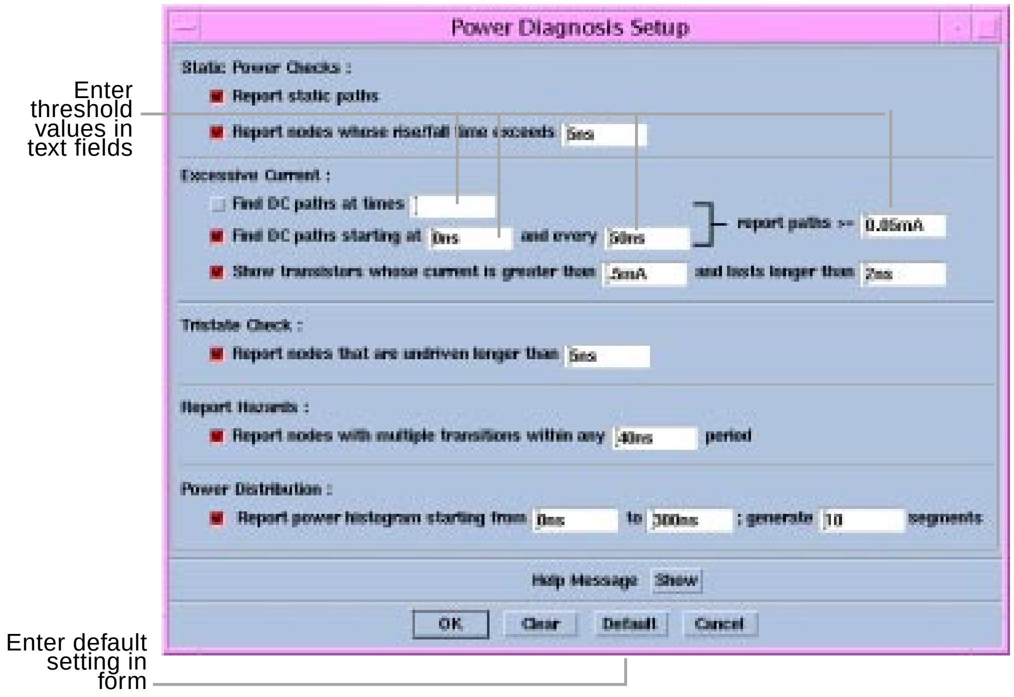
- **Saving:** After you've made your changes, click the **OK** button to save your settings and close the **Block Setup** form.
- **Clearing:** Click the **Clear** button to clear all information on the form.
- **Cancelling:** Click the **Cancel** button to close the form without saving your changes.

If you do not invoke the **Power Supply & Block Power Setup** form, default values are automatically assumed. The supply node voltages default to the values specified in the technology file. The power for the top-level blocks is automatically calculated if there are less than 100 blocks in the design.

NOTE: The configuration commands needed to calculate the power for the selected blocks are placed in a file called *supplyblock.cfg* in the Powermill GA run directory (by default, *powermill_pwga*) and are automatically applied to the run.

Power Diagnosis Setup

You can use the **Power Diagnosis Setup** form to set static power, excessive current, tristate, hazard, and power distribution checks for the simulation. PowerMill will check only those conditions you set (see Figure 6).


Enter threshold values in text fields
Enter default setting in form

Power Diagnosis Setup

Static Power Checks :

- [✓] Report static paths
- [✓] Report nodes whose rise/fall time exceeds 5ns

Excessive Current :

- [ ] Find DC paths at times
- [✓] Find DC paths starting at 0ns and every 50ns — report paths >= 0.05mA
- [✓] Show transistors whose current is greater than .5mA and lasts longer than 2ns

Tristate Check :

- [✓] Report nodes that are undriven longer than 5ns

Report Hazards :

- [✓] Report nodes with multiple transitions within any 40ns period

Power Distribution :

- [✓] Report power histogram starting from 0ns to 300ns ; generate 10 segments

Help Message Show

OK Clear Default Cancel

Figure 6 Power diagnosis setup form

The following steps provide the basic procedure for using the **Power Diagnostics Setup** form.

1 Click on the **Power Diagnosis** button in the **Simulation Setup** form.

The **Power Diagnostics Setup** form will appear.

2. Select the checks you want by clicking the appropriate toggle buttons located on the left side of the form.

   The check is active when the button is red and contains a black check mark. Many checks require you to enter threshold values in text fields.

3. Click the **Default** button at the bottom of the form to select the default power diagnosis checks. The **Clear** button de-selects all checks.

Static Power Checks

There are two buttons you can click in the **Static Power Checks** section of the **Power Diagnosis Setup** form. These buttons have the following functions:

- **Report static DC paths**: checks for circuit topologies that are likely to cause unwanted power consumption. For example, the settings in Figure 9 will catch circuits with no paths to VDD or GND.
- **Report node whose rise/fall time exceeds...**: checks for nodes with rise/fall times that exceed a specified time.

Excessive Current

The **Excessive Current** section lets you specify frequency and current consumption thresholds for reporting wasted power paths from supply to ground. If you do not specify a simulation time, then paths are reported for all time points in the simulation.

You can also specify thresholds for reporting elements. This allows you to find violations that are likely to affect device and wire current consumption limits.

Tristate Check

The **Tristate Check** button checks for nodes that are undriven for a specified time. This allows you to keep track of nodes that leak above a specified voltage value.

Report Hazards

The **Report Hazards** button checks for nodes that have multiple transitions within a specified time period. This check is helpful for finding signal transitions that consume excessive power.

Power Distribution

The **Power Distribution** button creates a histogram of average and peak current for a particular time period. For example, if you specify a time window from 0 to 100 ns in four segments, then a set of histograms will be created for the following time periods: 0-25, 26-50, 51-75, and 76-100 ns.

Showing or Hiding Help Messages

You can view or hide help messages using the Show/Hide toggle button to the right of the **Help Messages** heading.

Saving or Clearing Checks

When you finish specifying the checks, click the **OK** button. This saves your settings and closes the **Power Diagnosis Setup** form.

If you do not want to save your changes, click the **Cancel** button to close the form without saving.

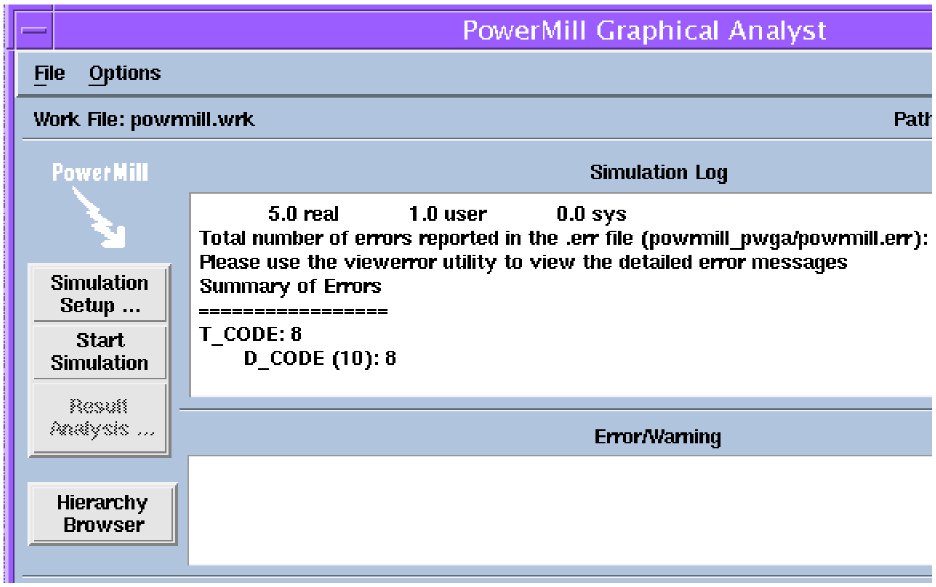
If you do not use the **Power Diagnosis Setup** form, default values are automatically assumed. These values are visible when you bring up the form.

NOTE: The configuration commands needed to perform the checks are placed in a file called *diag.cfg* in the Powermill GA run directory (by default, *powermill_pwga*) and are automatically applied to the run.

Running a Simulation

To start the simulator, click the **Start Simulation** button located on the upper left side of the **PowerMill GA** main window. See Chapter 2 for information on what you need to run a basic simulation.

As the simulation progresses, log and warning messages appear in the **Simulation Log** and **Error/Warning** fields of the main window (see Figure 7).



```
        5.0 real         1.0 user         0.0 sys
Total number of errors reported in the .err file (powmill_pwga/powmill.err):
Please use the viewerror utility to view the detailed error messages
Summary of Errors
=================
T_CODE: 8
    D_CODE (10): 8
```

Figure 7 Example log information in main window

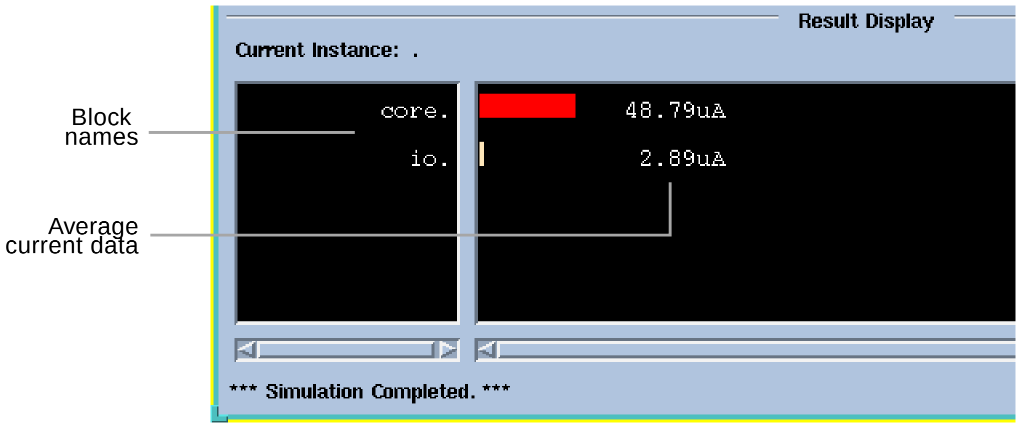
To stop the simulation at any time, click the **Stop Simulation** button; the **Start Simulation** button becomes the **Stop**

Simulation button once simulation has begun and reverts to **Start Simulation** when it is completed.

Analyzing a Circuit

Power Consumption

When a simulation is complete, average current information for the first level of the circuit hierarchy appears in the **Result Display** area of the **PowerMill GA** main window (see Figure 8).



Current Instance: .

```
core.   48.79uA
io.      2.89uA
```

*** Simulation Completed. ***

Figure 8 Average current information

In the example, the left column of the **Result Display** area shows the block names. The right column shows the average current consumption.

Sort By Menu

You can view other power consumption information using the **Sort By** menu located on the top right side of the **Result Display** area (see Figure 9).

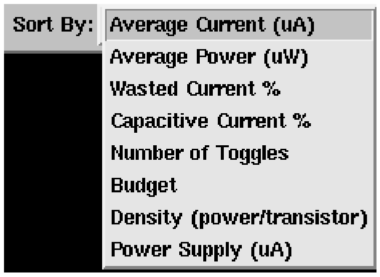


Figure 9 Sort by menu

Select the following menu items to get different types of power information:

- **Average Power Information**—select to view average power information.
- **Wasted Current %**—select to view information on short circuit current.
- **Capacitive Current Information**—select to view capacitive current information.
- **Budget—**select to view power budget information.

  If you specified a budget in the **Power Supply &Block Setup** form, the blocks are sorted according to how much they exceed the budget. Blocks displayed in red exceed the budget. Blocks displayed in yellow are under budget.
- **Density**—select to view density information.
- **Power Supply**—select to view the power consumption for each power supply node.

Hierarchical View of Power Consumption

You can view power consumption information for lower level blocks if they are specified in the **Power Supply and Block Setup** form.

To push down into a hierarchical block, double click with the left mouse button on the desired block. To pop back up a level, double click on a lower-level block, with the right mouse button.

Results Analysis

To continue analyzing the circuit, click the **Results Analysis** button located on the left side of the **PowerMill GA** main window. The **Results Analysis** form will appear (see Figure 10).

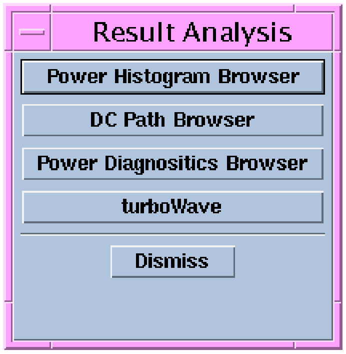


Figure 10 Result analysis form

To open a particular results analysis tool in the **Result Analysis** form, click the appropriate button.

Power Histogram

The **Power Histogram** allows you to hierarchically view power consumption information for selected blocks. To open the **Power**

Histogram, click the **Power Histogram** button in the **Result Analysis** form (see Figure 11).

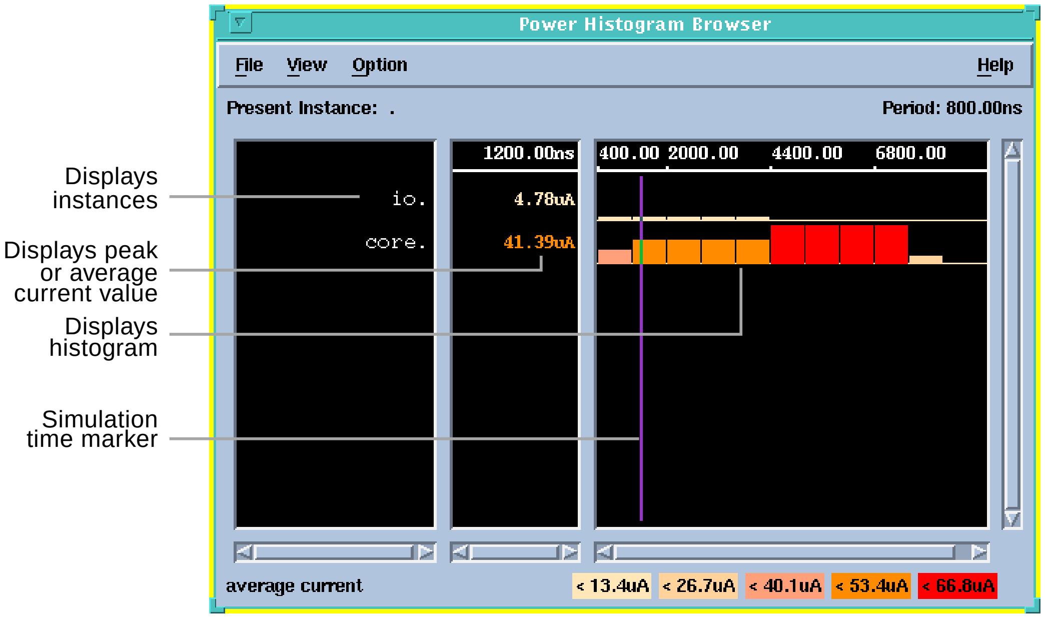


Figure 11 Power histogram browser

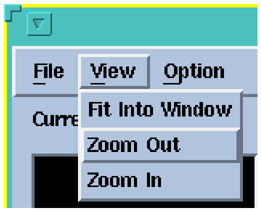
File Menu

After you've completed your hierarchical power analysis, you can close the **Power Histogram Browser** by selecting **Close Power Histogram Browser** from the **File** menu.

View Menu

You can analyze power consumption data in the **Power Histogram Browser** from a variety of views, including zoomed-out and zoomed-in views, and block and supply net views.

To see a zoomed-out view of the display, select **Zoom Out** from the **View** menu (see Figure 12).



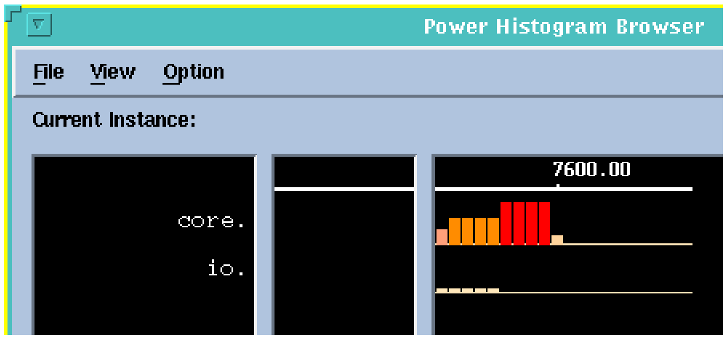


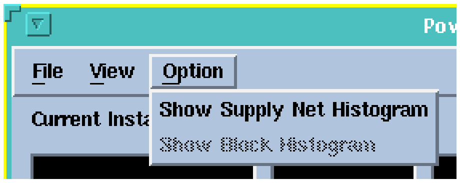
Figure 12 Zoom out view of power histogram browser

Select **Zoom In** from the **View** menu to return to the original view.

To fit the entire power consumption display into the window, select **Fit Into Window** from the **View** menu.

Options Menu

The **Options** menu lets you view either the supply net or block net histogram.To see the supply net histogram, select **Options→Show Supply Net Histogram** (see Figure 13).



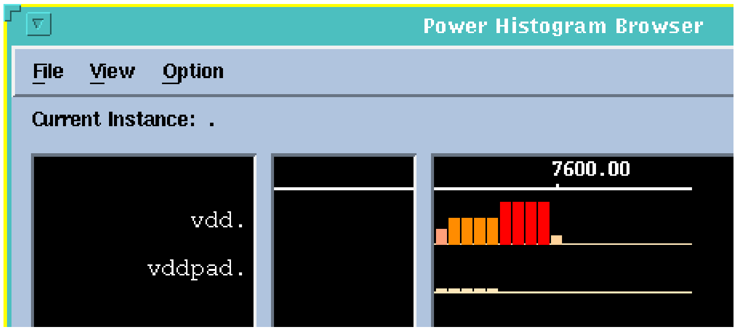


Figure 13 Supply net histogram

To return to the block histogram, select **Options→Show Block Histogram.**

Using the Histogram

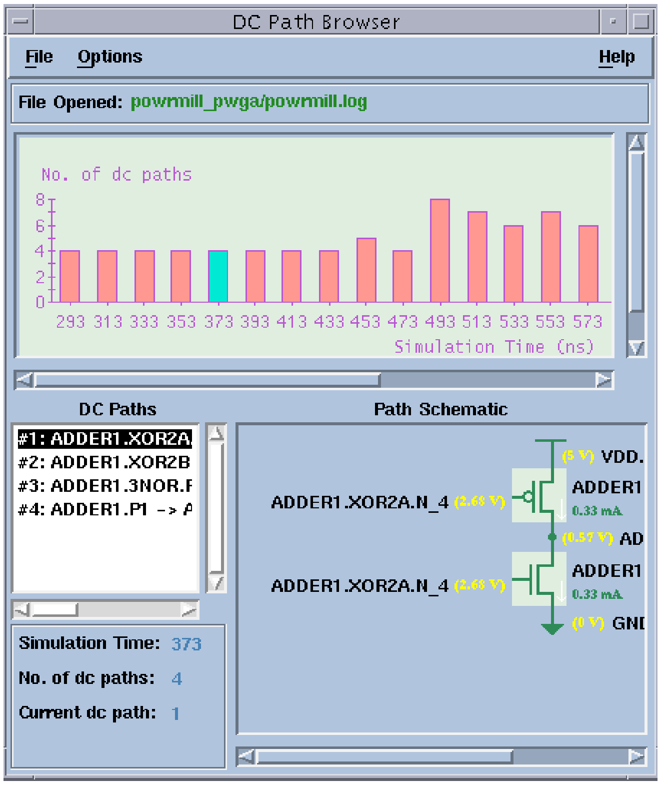
You can view the average or peak current for each time segment displayed in the histogram.To view the average current for a particular time segment, click your left mouse button on a time segment block in the histogram. A time marker, along with the average current for the selected segment, will appear.

To view the peak current for a segment, use your middle mouse button to click on a segment, or move the marker to another segment.

You can access the histogram for lower level blocks if they have been specified in the **Supply and Block Setup** form). To do this, double click on a block with your left mouse button. To come up a level, double click on a block with the right mouse.

DC Path Browser

The **DC Path Browser** graphically displays transistors from DC paths at predefined intervals. To open the **DC Path Browser**, click the **DC Path Browser** button in the **Result Analysis** form (see Figure 14).



File Opened: powmill_pwga/powmill.log

#1: ADDER1.XOR2A
#2: ADDER1.XOR2B
#3: ADDER1.3NOR.F
#4: ADDER1.P1 -> A

Simulation Time: 373
No. of dc paths: 4
Current dc path: 1

Figure 14 DC path browser

File Menu

After you've completed your DC path analysis, you can close the **DC Path Browser** by selecting **Close DC Path Browser** from the **File** menu.

Option Menu

Select **Option**→**Search** to bring up the **Search DC Paths** form. This form allows you to specify a particular time period of the simulation for viewing DC paths. This is useful if you are interested in seeing DC path for a particular time of the simulation, or if you cannot display all the DC paths in a single window (see Figure 15).

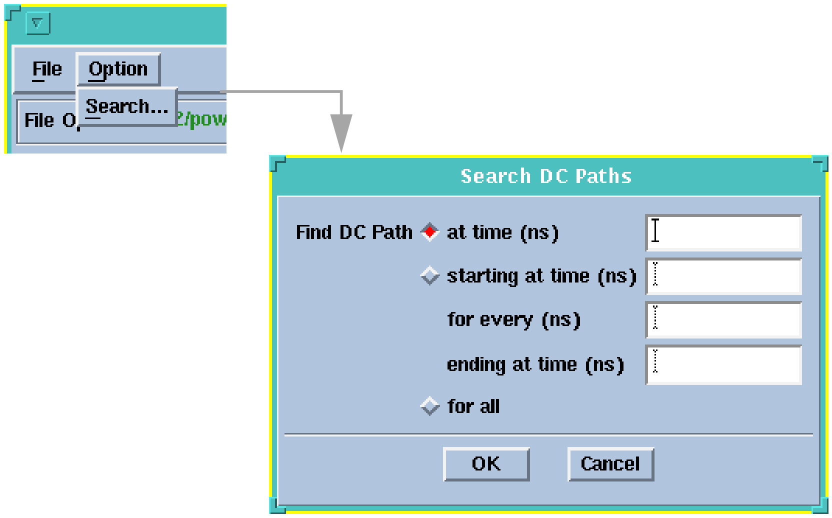
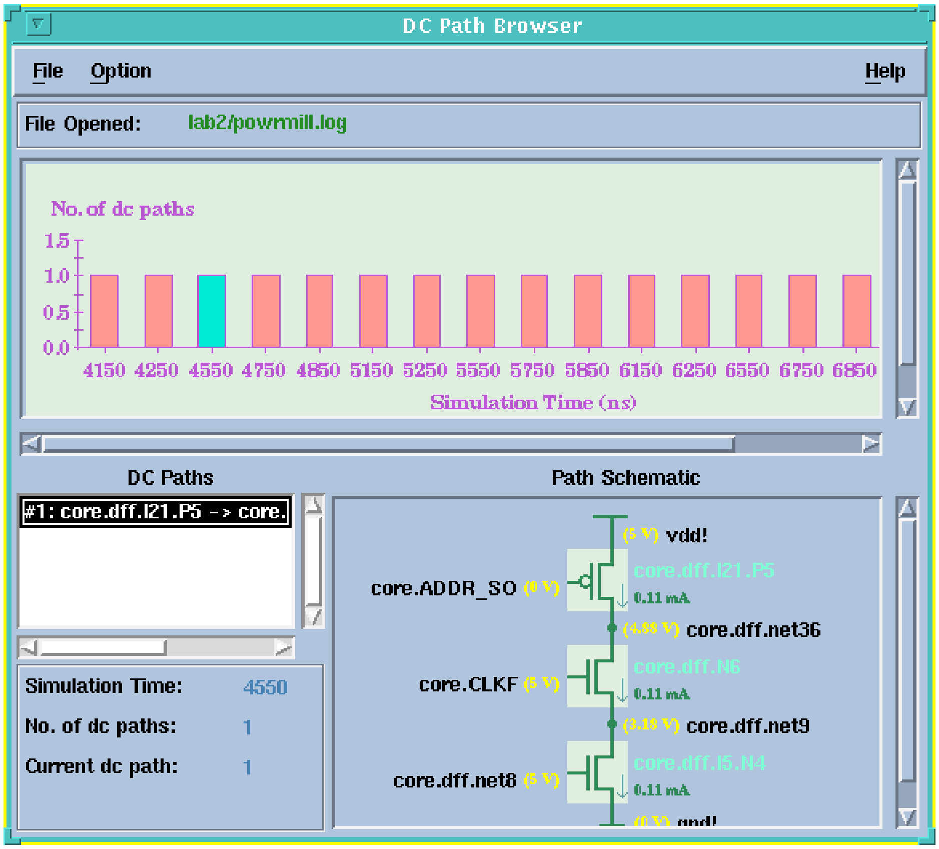


Figure 15 Search DC paths form

Analyzing DC Paths

The **DC Path Browser** automatically loads the *.dcpath* file for the current run. A series of time blocks are displayed at the top of the window. You can view detailed DC path information for a particular time period by clicking on any time segment (see Figure 16).



File Opened: lab2/powrmill.log

#1: core.dff.I21.P5 -> core.

Simulation Time:	4550
No. of dc paths:	1
Current dc path:	1

Figure 16 Detail view of DC path browser

When you've clicked a particular time segment, all the DC paths that occurred at that time are displayed in the **DC Paths** text field located at the bottom left corner of the window. By default, the first path is displayed in the **Path Schematic** area at the bottom right corner of the browser.

To view a particular path, click a path name in the **DC Paths** text field. The schematic for the path you selected is displayed in the **Path Schematic** area.

Standalone Version of DC Path Browser

The **DC Path Browser** can be run as a standalone tool. To invoke this version, use the **dcpviewer** command.

EXAMPLE:

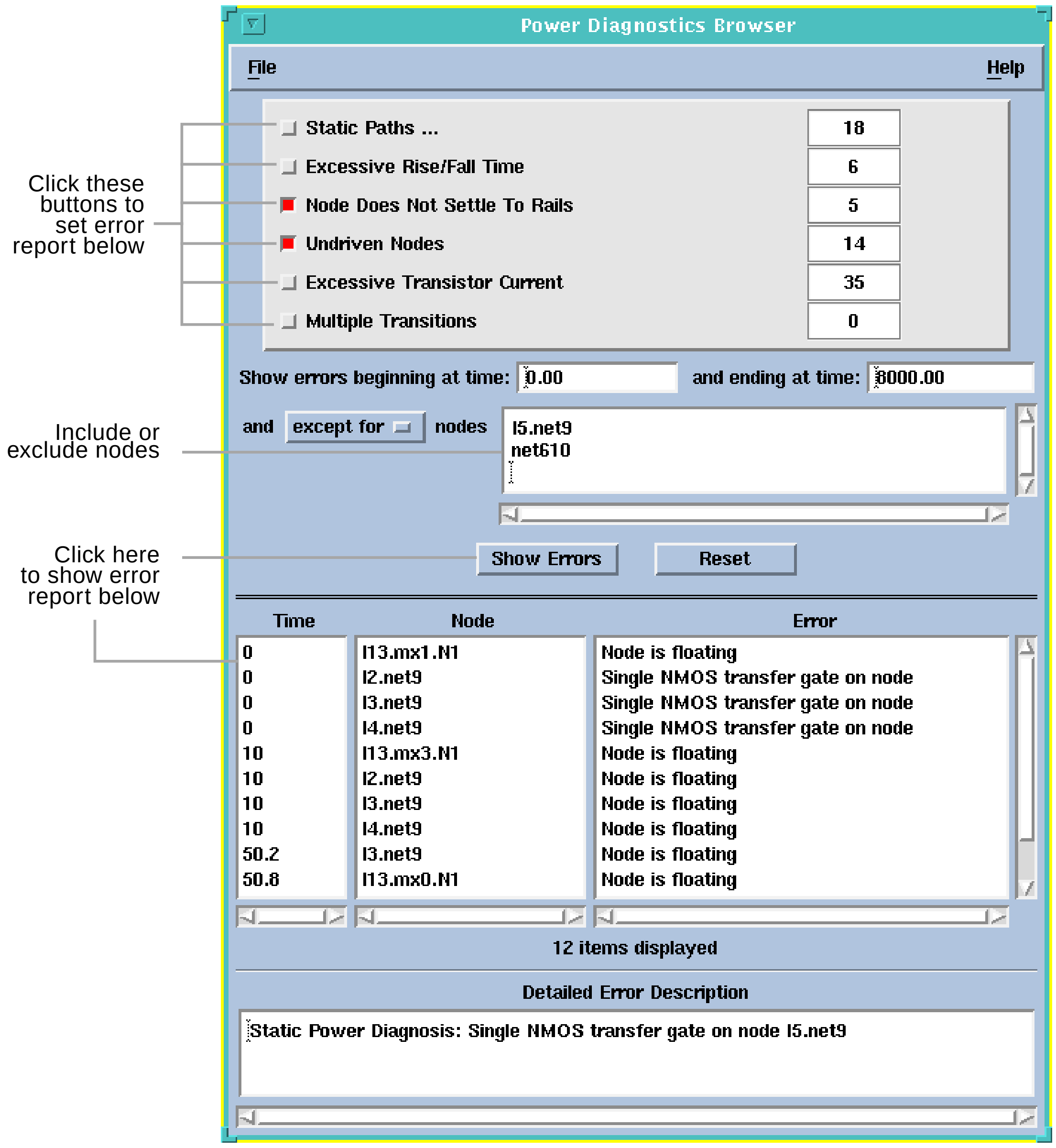
```
dcpviewer .dcpth_filename
```

The **DC Path Browser** will appear and display the DC paths contained in the particular *.dcpath* file you specified. In standalone mode, you can open any Powermill *.dcpath* file by selecting **File→Open Log File**.

Power Diagnostics Browser

You can use the **Power Diagnostics Browser** to graphically view and sort diagnostic information, including static configuration checks, multiple toggles, and dynamic rise/fall times.

To load the **Power Diagnostics Browser**, click the **Power Diagnostics Browser** button in the **Result Analysis** form (see Figure 17).


Click these buttons to set error report below
Include or exclude nodes
Click here to show error report below
Power Diagnostics Browser
File
Help
Show Errors
Reset

Static Paths ...	18
Excessive Rise/Fall Time	6
🟥 Node Does Not Settle To Rails	5
🟥 Undriven Nodes	14
Excessive Transistor Current	35
Multiple Transitions	0

Show errors beginning at time: 0.00 and ending at time: 8000.00

and except for nodes I5.net9 net610

Time	Node	Error
0	I13.mx1.N1	Node is floating
0	I2.net9	Single NMOS transfer gate on node
0	I3.net9	Single NMOS transfer gate on node
0	I4.net9	Single NMOS transfer gate on node
10	I13.mx3.N1	Node is floating
10	I2.net9	Node is floating
10	I3.net9	Node is floating
10	I4.net9	Node is floating
50.2	I3.net9	Node is floating
50.8	I13.mx0.N1	Node is floating

12 items displayed

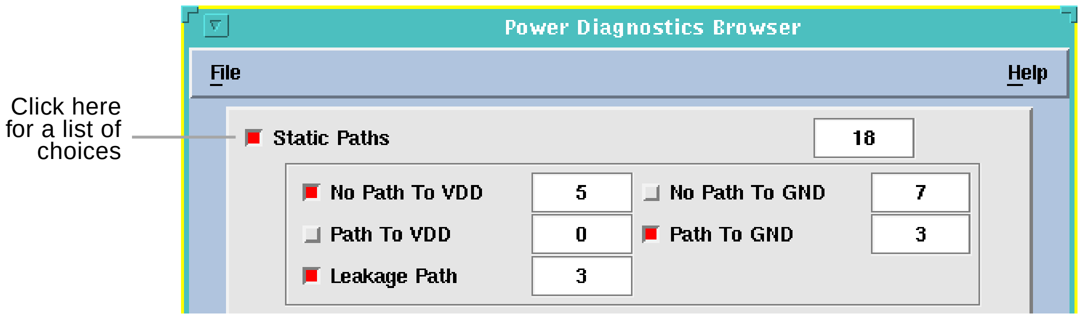
Detailed Error Description

Static Power Diagnosis: Single NMOS transfer gate on node I5.net9

Figure 17 Power diagnostics browser

Selecting Checks

The numbers that appear in the upper part of the browser indicate the number of messages in each error category. Select the type of errors you want to view by clicking the appropriate buttons at the top of the form. The button will turn red when a check is active (see Figure 18).



🟥 Static Paths	18		
🟥 No Path To VDD	5	☐ No Path To GND	7
☐ Path To VDD	0	🟥 Path To GND	3
🟥 Leakage Path	3		

Figure 18 Selecting static paths

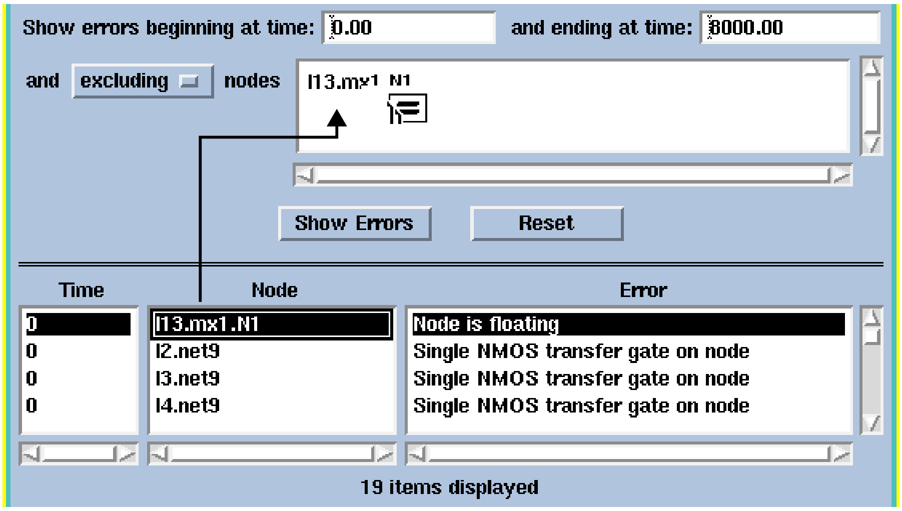
Use the following buttons related to selecting checks:

- **Static Checks:** causes additional selections to appear.
- **Show Errors:** shows a list of errors corresponding to the error types you selected.

  Select a particular error to get a detailed information on that error at the bottom of the browser.
- **Reset:** clears your selections.

You can use your middle mouse button to drag and drop node names from the **Error List** to other parts of the window. You can also drag and drop node names from other sources in the

PowerMill GA, such as the **Hierarchy Browser** (see Figure 19).


Show errors beginning at time: 0.00 and ending at time: 8000.00

and excluding nodes I13.mx1 N1

Show Errors | Reset

Time	Node	Error
0	I13.mx1.N1	Node is floating
0	I2.net9	Single NMOS transfer gate on node
0	I3.net9	Single NMOS transfer gate on node
0	I4.net9	Single NMOS transfer gate on node

19 items displayed

Figure 19 Dragging and dropping a node name

The **Power Diagnostics Browser** provides wild card support. For example, if you want to see the errors on nodes in block A, you can select the following:

```
only on nodes:

A

or A.

or  A*.
```

By default, selected errors on all nodes are displayed. However, to help in filtering the errors, you can use wild cards to filter nodes for particular blocks. For example, you can select **only on** nodes A or A**.** if you only want to see the errors on the nodes in block A. Another option is to drag the block A from the **Hierarchy Browser**.

To view detailed information, select an **Error** in the display area. A detailed description is printed in the **Detailed Error Description** window.

File Menu

After you have completed your diagnostic information analysis, you can close the **Power Diagnostic Browser** by selecting **File→Close Power Diagnostics Browser**.

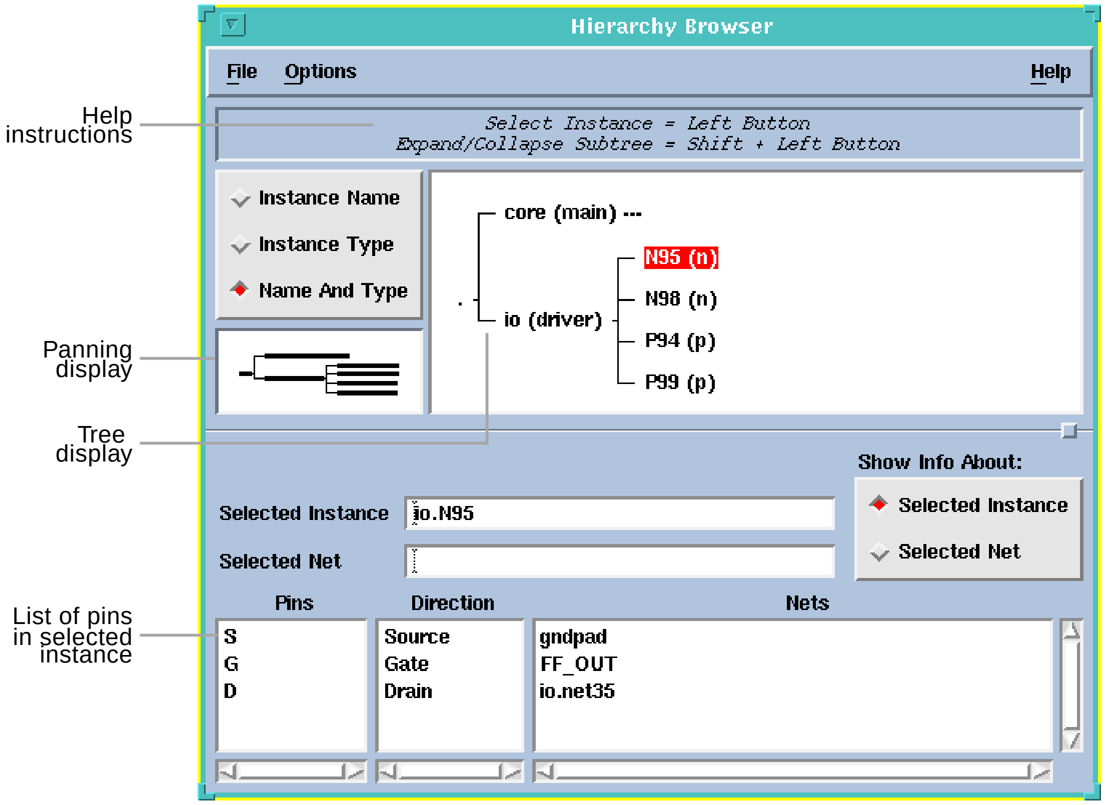
Waveform Tool

The Powermill Graphical Analyst supports two waveform tools: Simwave and turboWave. The default tool is turboWave.

Use the **Options** menu of the **PowerMill GA** main window to select a particular waveform tool. Based on your selection, the **Results Analysis** window will display and invoke either turboWave or Simwave.

Hierarchy Browser

The **Hierarchy Browser** provides you with a visual display of the netlist hierarchy. To open the **Hierarchy Browser**, click the **Hierarchy Browser** button located at the middle left side of the **PowerMill GA** main window (see Figure 20).



Select Instance = Left Button
Expand/Collapse Subtree = Shift + Left Button

Selected Instance: io.N95

Selected Net:

Pins	Direction	Nets
S	Source	gndpad
G	Gate	FF_OUT
D	Drain	io.net35

Figure 20 Hierarchy browser

File Menu

After you are done using the **Hierarchy Browser**, you can close it by selecting **File**→**Close Hierarchy Browser** located at the top left corner of the window.

Hierarchy Display

The **Hierarchy Browser** contains the following display areas:

- **Panning display:** located on the left side of the window, this shows you what portion of the tree is currently being displayed in the tree display.
- **Tree display:** located on the upper right side of the window, this shows a portion of the netlist's instance hierarchy.

Panning Display

The panning display shows the layout of the tree. The rectangle in the tree layout indicates the portion of the tree that is currently visible in the tree display. You can update the portion of the tree displayed by clicking on the rectangle and dragging it.

Tree Display

Initially, the hierarchy in the tree display consists only of top-level design components. You can expand portions of the instance tree by pressing the shift key and clicking your left mouse button on instances. You can collapse the hierarchy by holding down the shift key and clicking a previously expanded instance.

When you click on the instance name, it becomes red. Pin information for the selected instance is displayed in the text fields at the bottom of the **Hierarchy Browser**. The instance name is also displayed in the **Selected Instance** text field.

You can choose to view the tree display by instance name or type, or both by clicking the **Instance Name**, **Instance Type**, or **Name and Type** buttons located at the top left corner of the **Hierarchy Browser**.

You can view the attributes of a net in the tree display by clicking the **Selected Net** button located at the bottom right corner of the window, then selecting a net name from the list below.

Show Info

The data displayed in the instance information fields depends on the button you click in the **Show Info About** box. If you click the **Selected Instance** button, a list of pins on the selected instance is displayed, including direction information, and the name of the net the pins connect to. If you click the **Selected Net** button, the displayed information includes instances connected to the selected net, the instance pin connecting to the net, and the pin directions.

Options

The **Options** menu, located at the top of the **Hierarchy Browser**, lets you change how information is displayed.

If you select the dashed line at the top of the **Options** menu, the menu becomes a dialog box.

Show Internal Nets

Normally, when you click on an instance name in the tree display, a list of nets connected to the instance's external pins appears at the bottom of the display. If you select **Options→Show internal Nets** from the menu, the instance's internal nets will also appear in the list.

Show Instructions

Select **Show Instructions** from the **Options** menu to display instructions for using the **Hierarchy Browse**r. This is a toggle option; clicking it will either display or remove the instructions. The instructions appear at the top of the window

Ordering Instance Information

You can sort the data displayed in the text fields at the bottom of the **Hierarchy Browser** three different ways. Select one of the following buttons from the **Options** menu:

- **Pin Name:** lists information sorted by pin name.
- **Pin Direction**: lists information by the pin direction.
- **Net Name**: lists information sorted by net name.

Set Expansion Threshold

You can set the maximum number of child cells that the browser will display when you are expanding an instance in the tree display into its child components (see Figure 21).

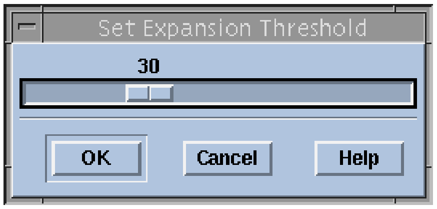


Figure 21 Set expansion threshold form

Procedure

1 Select **Options→Set Expansion Threshold**.

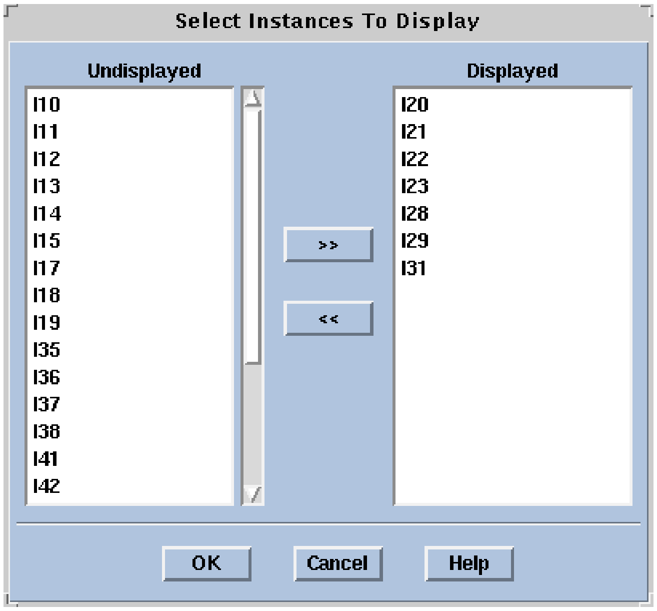
The **Set Expansion Threshold** form will appear.

2 Click on the sliding bar and move it until the appropriate setting appears above the bar.

3 Click **OK** when you are done adjusting the bar.

Select Instances to Display

If an instance contains more child cells than the expansion threshold, the **Select Instances to Display** form will appear (see Figure 22).



Undisplayed	Displayed
I10	I20
I11	I21
I12	I22
I13	I23
I14	I28
I15	I29
I17	I31
I18	
I19	
I35	
I36	
I37	
I38	
I41	
I42	

Figure 22 Select instances to display window

This form allows you to choose the child cells you want to display in the instance tree.

Choosing Child Cells to Display in the Instance Tree

1. Select the child cells you want to display.
2. Press the >> button to move them to the **Displayed** column.
3. If you do not want child cells to display, select them from the **Displayed** column and press the << button.

   The child cells will move to the **Undisplayed** column.

Choosing Multiple Instances at one Time

1 Select an instance.

2 Hold down the Shift key and click on the last selection you want in the list.

   All the names from the first selection to the last are selected.

   To de-select an instance, press the control key and click on an instance.

When the **Select Instances to Display** form is complete, the selected child cells will appear in the tree display. A file cabinet icon represents those child cells not displayed. To expand the file cabinet, press the Shift key and click your left mouse button. This re-opens the selection form so you can add or remove child cells from the display.

If you select an instance inside a file cabinet, it will appear in the tree display. If the display of the instance's children exceeds the expansion threshold, the **Select Instances to Display** window will appear.

Save Options

You can save the current option states by selecting **Save Options** from the **Options** menu. This causes the **Hierarchy Browser** to use the current option states the next time the current design is opened.

Displaying Connectivity

To start tracing a path, click on an instance name in the tree display hierarchy. Then, select a pin from the list at the bottom of the browser.

To trace a path, you must select both an instance and a pin. You can select only one instance and one pin at a time.

When you select a pin, the parent, children, and sibling instances connected in the net to the selected instance are highlighted in blue in the tree display (see Figure 23).

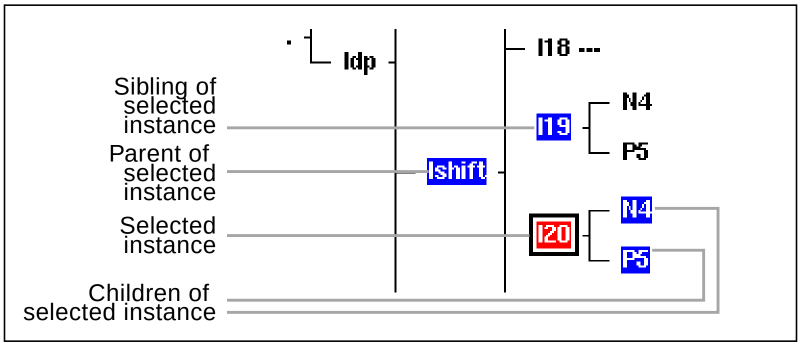

Idp
I18 ---
Sibling of selected instance
Parent of selected instance
Selected instance
Children of selected instance
Ishift
I19
N4
P5
I20
N4
P5

Figure 23 Example of a selected instance and related instances

If you click another connected instance, for example a sibling instance, the hierarchy browser display will change to reflect the new hierarchy context. Again, only the parent, children, and sibling instances of the selected instance will be highlighted in blue (see Figure 24.

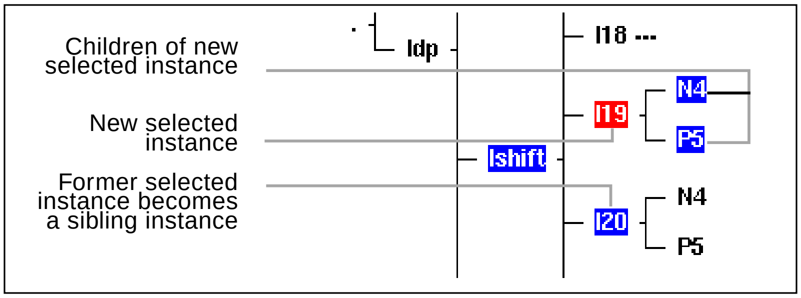

Idp
I18 ---
Children of new selected instance
New selected instance
Former selected instance becomes a sibling instance
I19
N4
P5
Ishift
I20
N4
P5

Figure 24 Context shift after selecting a related instance

Drag and Drop

You can drag and drop text from the panning display area or the instance information fields located at the bottom of the **Hierarchy Browser**.

1 Click the middle mouse button on the text you want to drag, then move the mouse (cursor).

2 Release the middle mouse button when the cursor is in the appropriate position.

The text will appear in the new location.

Chapter 8

PowerMill Graphical Analyst Tutorial

Overview

This chapter gives you step-by-step instructions for running the PowerMill Graphical Analyst tutorial. The following tasks are covered:

- Getting the Input Files
- Starting the Graphical Analyst
- Setting Up for a Simulation
- Running the Simulation
- Analyzing Results

Getting the Input Files

Before you can run the graphical analyst tutorial in this chapter, you need to copy the required input files to your current working directory. The following procedure shows you how to do this.

1. Set the correct path to which the EPIC_HOME environment variable is set. If you're not sure of the path, you can ask your system administrator.
2. Copy (recursively) the files from *$EPIC_HOME/tutorials/pwga* to your local working directory. The files in this directory are needed for this tutorial.

   ```
   cp -R $EPIC_HOME/tutorials/pwga .
   ```

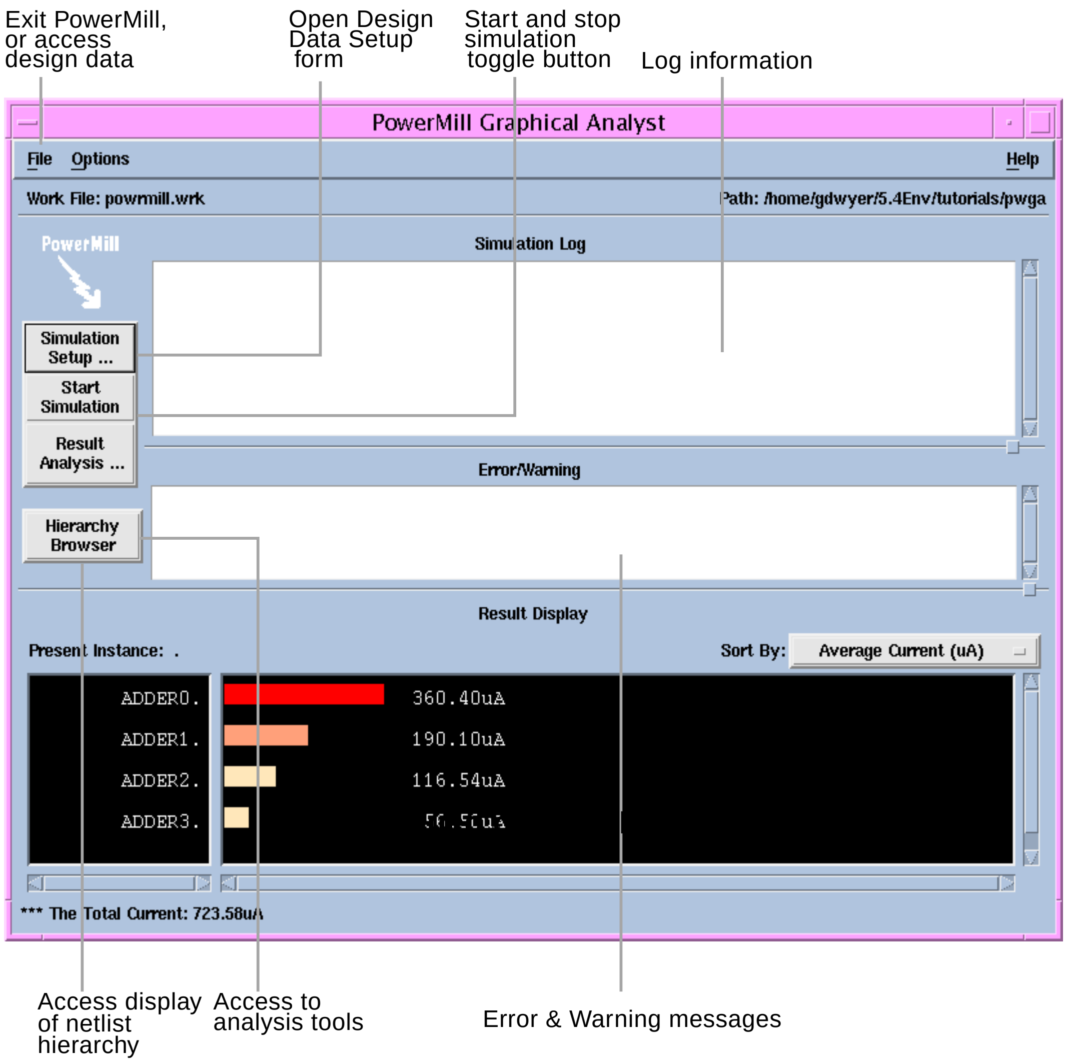
   This command copies *pwga* directory into the current directory (the period tells the **cp** command to copy to the current directory).

Starting the Graphical Analyst

Start the Graphical Analyst by typing one of the following commands at the UNIX prompt in your *pwga* directory:

```
pwga
powrmill -ga
```

The **Powermill GA** main window appears (see Figure 1).



Work File: powrmill.wrk

Path: /home/gdwyer/5.4Env/tutorials/pwga

Present Instance: .

Sort By: Average Current (uA)

```
ADDER0.   360.40uA
ADDER1.   190.10uA
ADDER2.   116.54uA
ADDER3.    56.56uA
```

*** The Total Current: 723.58uA

Figure 1 PowerMill GA main window

Setting Up for a Simulation

Your next set of steps involves specifying the design data, setting up multiple supply nodes, and specifying the blocks you want to simulate.

First, click the **Simulation Setup** button to bring up a dialog box that lists a set of simulation setup forms (see Figure 2).

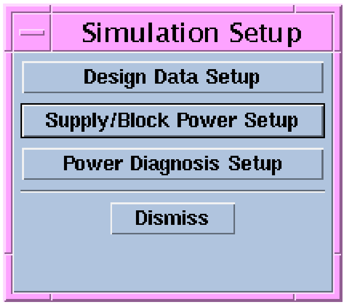


Figure 2 Simulation setup form

Specifying Design Data

To start specifying design data, click the **Design Data Setup** button in the **Simulation Setup** form.

The **Design Data Setup** form appears. Enter the following information either manually or using the **Browse...** button when applicable:

Work File: `powrmill.wrk`

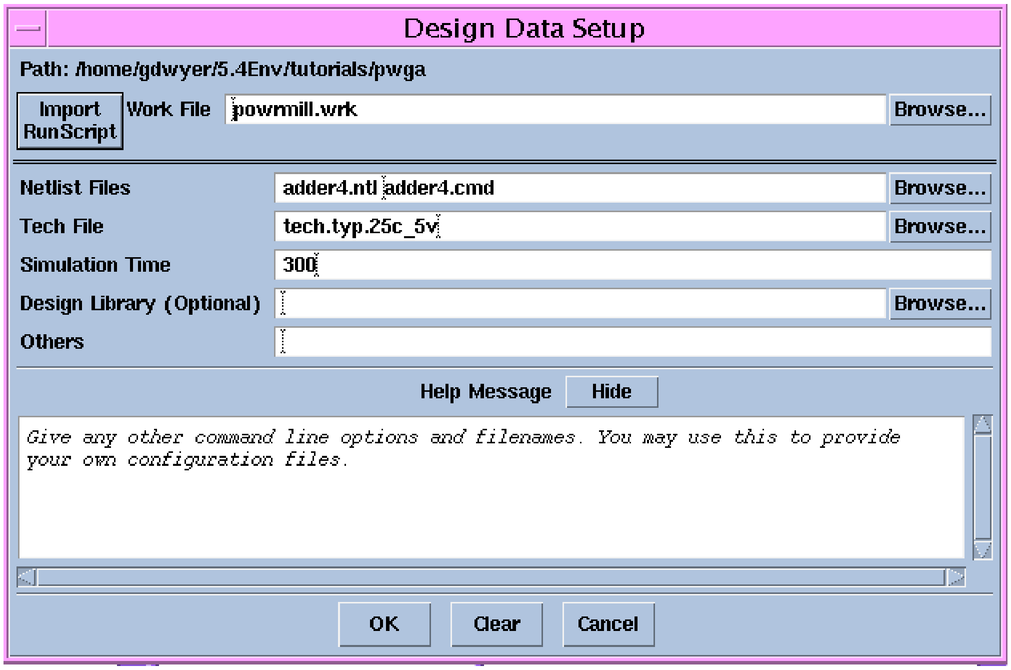
Netlist Files: `adder4.ntl adder4.cmd`

Tech File: `tech.typ.25c_5v`

Simulation Time: `300`

Figure 3 shows the completed form.

To save your changes and close the **Design Data Setup form**, click **OK**.



Path: /home/gdwyer/5.4Env/tutorials/pwga

Work File: powrmill.wrk

Netlist Files: adder4.ntl adder4.cmd

Tech File: tech.typ.25c_5v

Simulation Time: 300

Design Library (Optional):

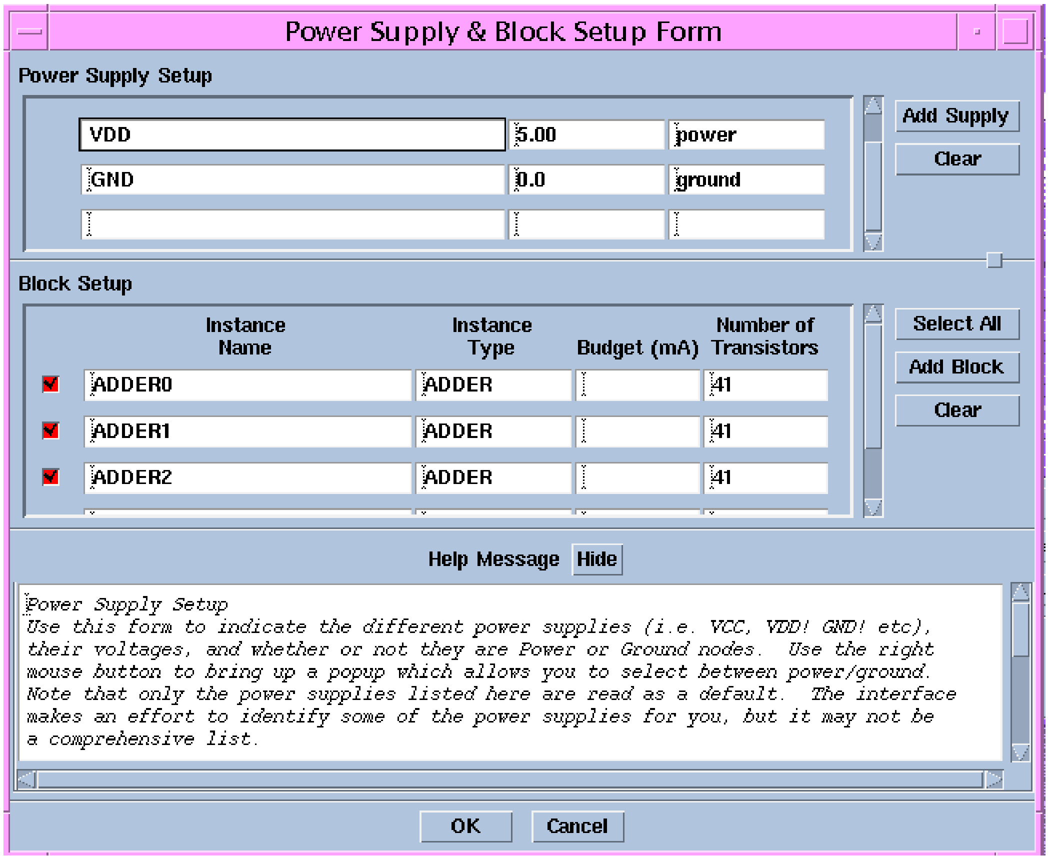
Others:

Give any other command line options and filenames. You may use this to provide your own configuration files.

Figure 3 Design data setup form

Setting Up the Power Supply and Blocks

The next step is to set up the power supplies and specify the blocks you want to use in the simulation. To do this, click the **Supply/Block Power Setup** button in the **Simulation Setup** dialog box. The **Power Supply & Block Setup** form will appear (see Figure 4).


Power Supply & Block Setup Form

Power Supply Setup

VDD	5.00	power
GND	0.0	ground

Block Setup

	Instance Name	Instance Type	Budget (mA)	Number of Transistors
✓	ADDER0	ADDER		41
✓	ADDER1	ADDER		41
✓	ADDER2	ADDER		41

Help Message

Power Supply Setup
Use this form to indicate the different power supplies (i.e. VCC, VDD! GND! etc), their voltages, and whether or not they are Power or Ground nodes. Use the right mouse button to bring up a popup which allows you to select between power/ground. Note that only the power supplies listed here are read as a default. The interface makes an effort to identify some of the power supplies for you, but it may not be a comprehensive list.

Figure 4 Power Supply & Block Setup form

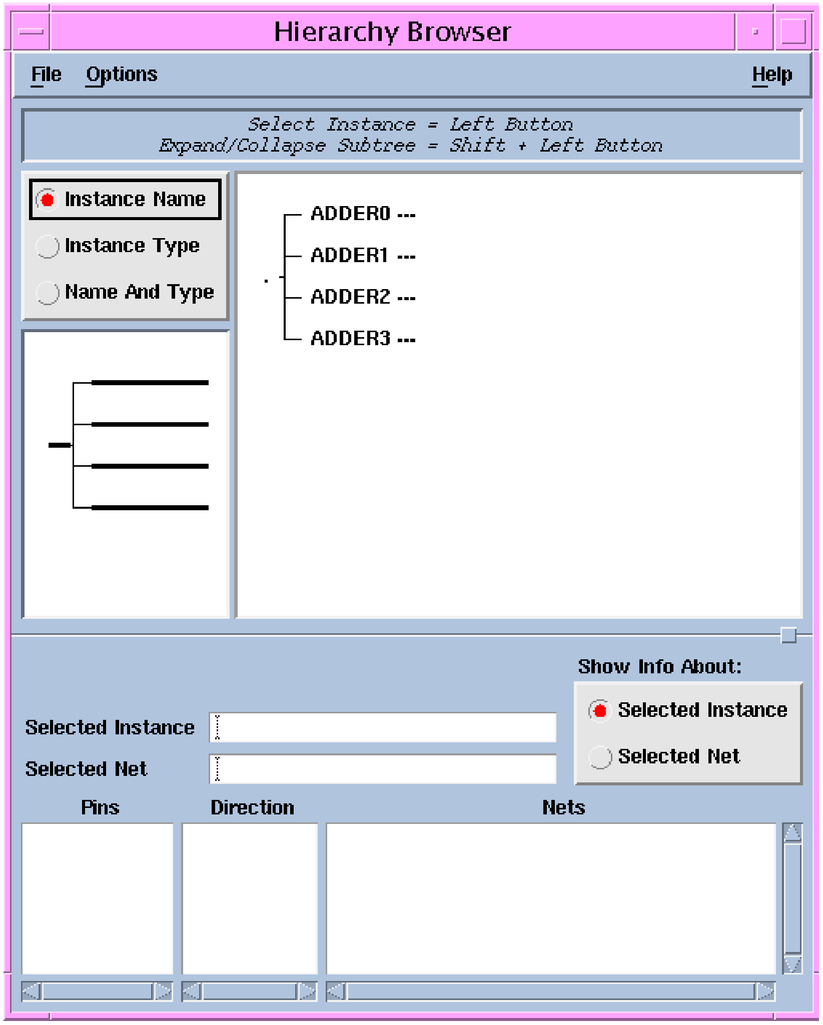
Normally at this point you would set up multiple power supplies. However, because the circuit used for this tutorial contains only one supply node, power supply set up is not necessary.

Your next step is to specify the blocks for which you want power consumption information. By default, all the blocks at the top level of the design are selected.

To get the power information for lower level blocks, you can add additional blocks by dragging and dropping them from the **Hierarchy Browser**.

First, bring up the **Hierarchy Browser** by clicking the **Hierarchy Browser** button in the **Powermill GA** main window.

The **Hierarchy Browser** appears (see Figure 5).



Select Instance = Left Button
Expand/Collapse Subtree = Shift + Left Button

[● Instance Name] Instance Type Name And Type

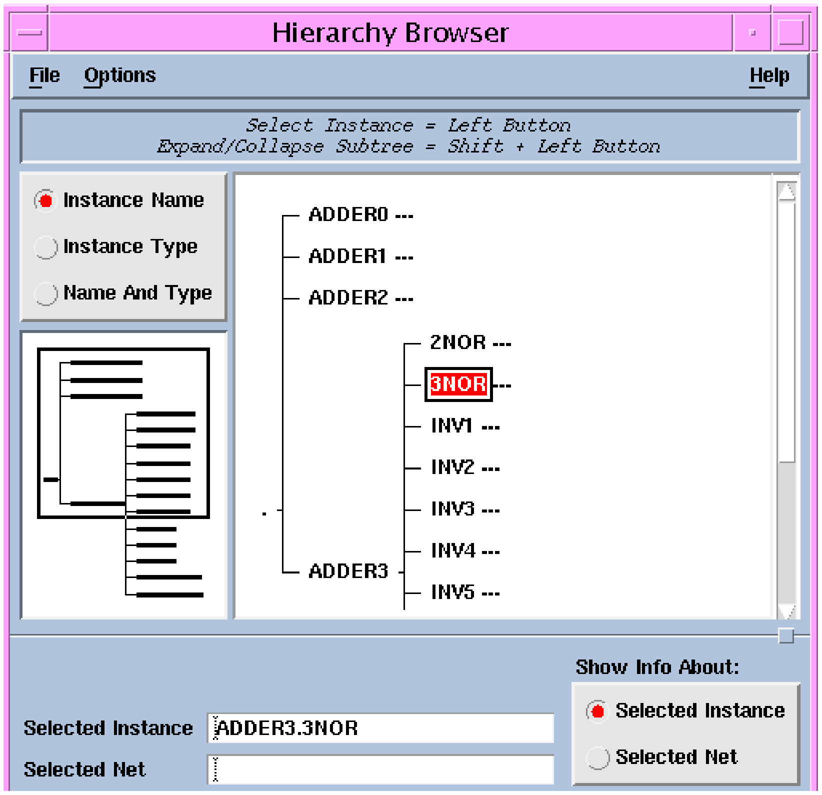
Selected Instance
Selected Net

Show Info About: [● Selected Instance] Selected Net

Figure 5 Hierarchy Browser

You can use the procedure outlined below to view the power consumption for lower-level blocks within the design hierarchy. This process shows you how to drag and drop blocks from the **Hierarchy Browser** to the **Power Supply & Block Setup** form.

1 In the top display area of the **Hierarchy Browser**, Hold down the Shift key and use the left mouse button to expand the hierarchy for the block identified as ADDER3. See Figure 6 for a sample.



Select Instance = Left Button
Expand/Collapse Subtree = Shift + Left Button

[● Instance Name] Instance Type Name And Type

Selected Instance ADDER3.3NOR
Selected Net

Show Info About: [● Selected Instance] Selected Net

Figure 6 Expanding the ADDER3 block

2 In the **Hierarchy Browse**r, select the 3NOR instance with the left mouse button. This highlights the instance.

3 Use the middle mouse button to drag it into the empty **Block Setup** field in the **Power Supply & Block Setup** form.

To add additional blocks, click the **Add Block** button in the form to create another empty field and use the drag and drop feature to include them.

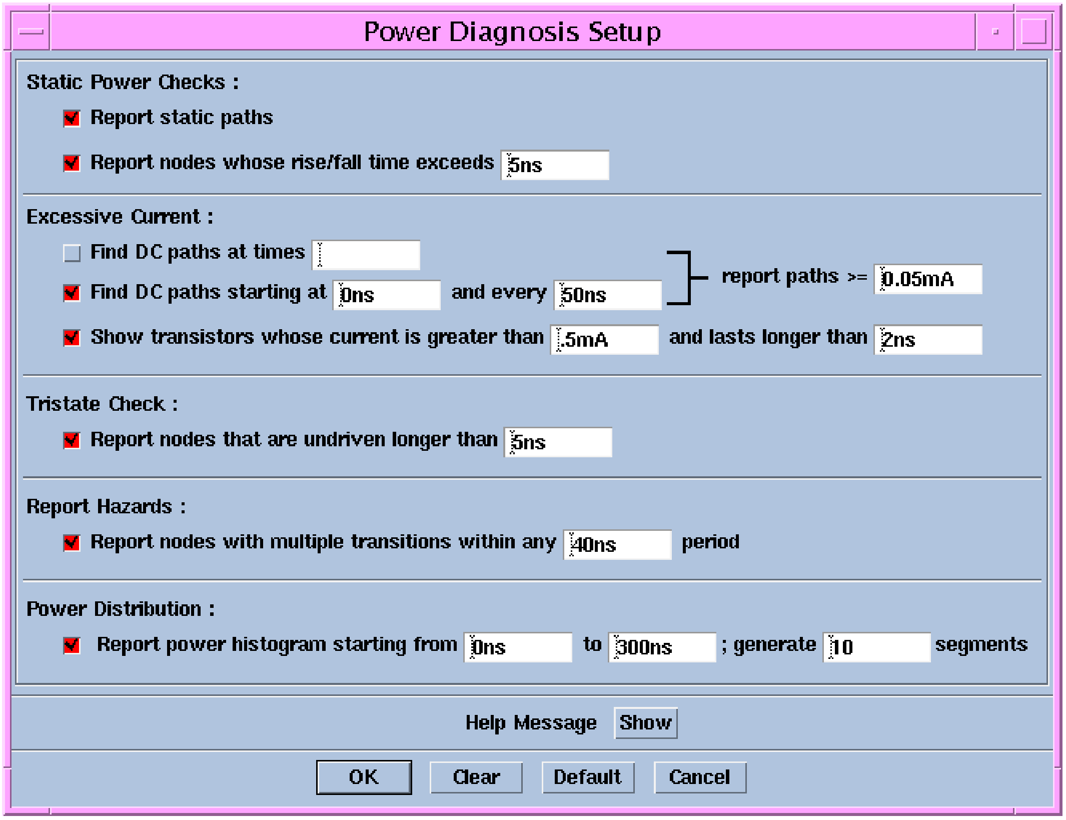
4 Select **File** → **Close Hierarchy Browser** to exit the **Hierarchy Browse**r.

5 Click **OK** in the **Power Supply & Block Setup** form to save your changes and exit the form.

Specifying Diagnosis Checks

PowerMill will check only those conditions you specify. To set the diagnosis checks, click the **Power Diagnosis Setup** button in the **Simulation Setup** form.

The **Power Diagnosis Setup** form appears (see Figure 7).


Power Diagnosis Setup

Static Power Checks :

- [✓] Report static paths
- [✓] Report nodes whose rise/fall time exceeds 5ns

Excessive Current :

- [ ] Find DC paths at times
- [✓] Find DC paths starting at 0ns and every 50ns

report paths >= 0.05mA

- [✓] Show transistors whose current is greater than .5mA and lasts longer than 2ns

Tristate Check :

- [✓] Report nodes that are undriven longer than 5ns

Report Hazards :

- [✓] Report nodes with multiple transitions within any 40ns period

Power Distribution :

- [✓] Report power histogram starting from 0ns to 300ns ; generate 10 segments

Figure 7 Power Diagnosis Setup form

By default, all the checks on the form are selected. However, in order to check for DC paths, you must specify the appropriate time segments.

Enter the following value in the **Static Power Checks** area:

Report nodes whose rise/fall time exceeds `.8ns`

Next, activate the **Excessive Current** checks area by clicking the toggle button located to the right of the **Find DC paths at times** line. Then, enter the values as shown below:

Find DC paths starting at `253ns` **and every** `20ns` **report paths** `>= .04mA`

Enter the following values in the **Report Hazards** area:

Report nodes with multiple transitions within any `1ns` **period.**

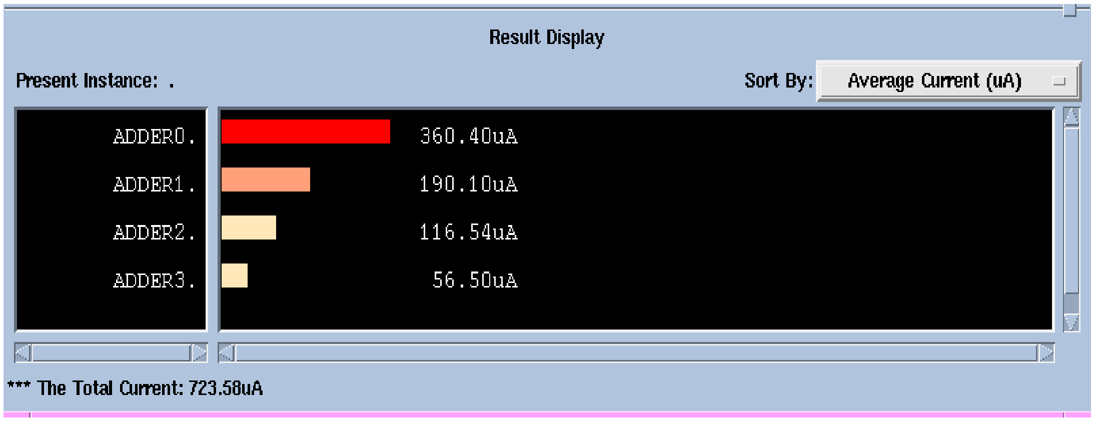
Click **OK** to save your changes and exit the **Power Diagnosis Setup** form. Then, click **Dismiss** in the **Simulation Setup** form. You are now finished with the set up process.

Running the Simulation

To run the simulation, click the **Start Simulation** button in the Powermill GA main window. Notice that the simulation messages are displayed in the **Simulation Log** text field. Note also that error and warning messages appear in the **Error/ Warning** text field.

When the simulation is finished, the **Result Display** area of the window displays power consumption of the first level of the design hierarchy.

Figure 8 shows the Results Display area of the **PowerMill GA** main window.



```
ADDER0.   360.40uA
ADDER1.   190.10uA
ADDER2.   116.54uA
ADDER3.    56.50uA
```

*** The Total Current: 723.58uA

Figure 8 Results Display area of the PowerMill GA main window

Note that you can use the **Sort By** menu to select various display options. Select **Wasted Current %** from the menu and notice how the display changes.

Use your left mouse button to view the lower levels of the hierarchy.

Double click with the left mouse button on the ADDER3 block. Notice the power consumption for the NOR3 instance is displayed.

To go back up a level in the hierarchy, double-click with the right mouse button on the NOR3 instance. Notice that the top-level blocks are displayed.

Analyzing Results

The PowerMill Graphical Analyst provides you with several tools for viewing the results of a simulation. You can access these tools by clicking the **Result Analysis** button in the Powermill GA main window.

The **Result Analysis** form appears (see Figure 9).

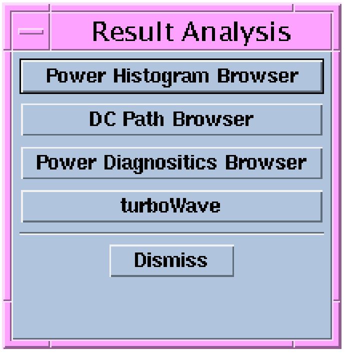


Figure 9 Result Analysis form

Viewing Histogram Information

First, you'll view a hierarchical display of power consumption information for selected blocks. To do this, click the **Power Histogram Browse** button in the **Result Analysis** form.

The **Power Histogram Browser** will appear (see Figure 10).

This tool displays the per cycle current consumption for the blocks you previously selected in the **Power Diagnostics Setup** form. In this case, 30 ns was specified.

To view the average current consumption per block and period, use your left mouse button to click any block in the display. Use your middle mouse button to click on a block and view the peak current consumption, per block and period.

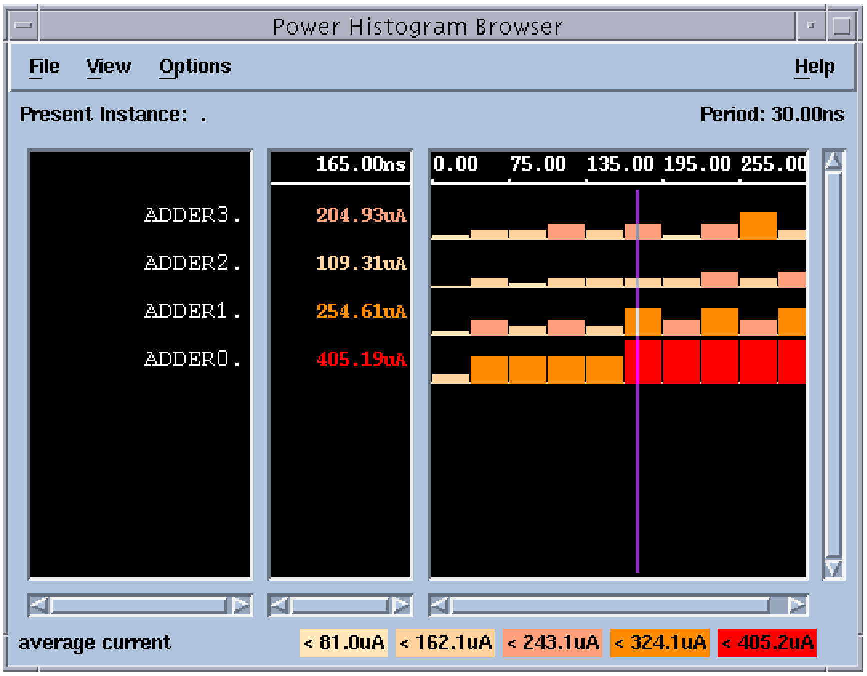
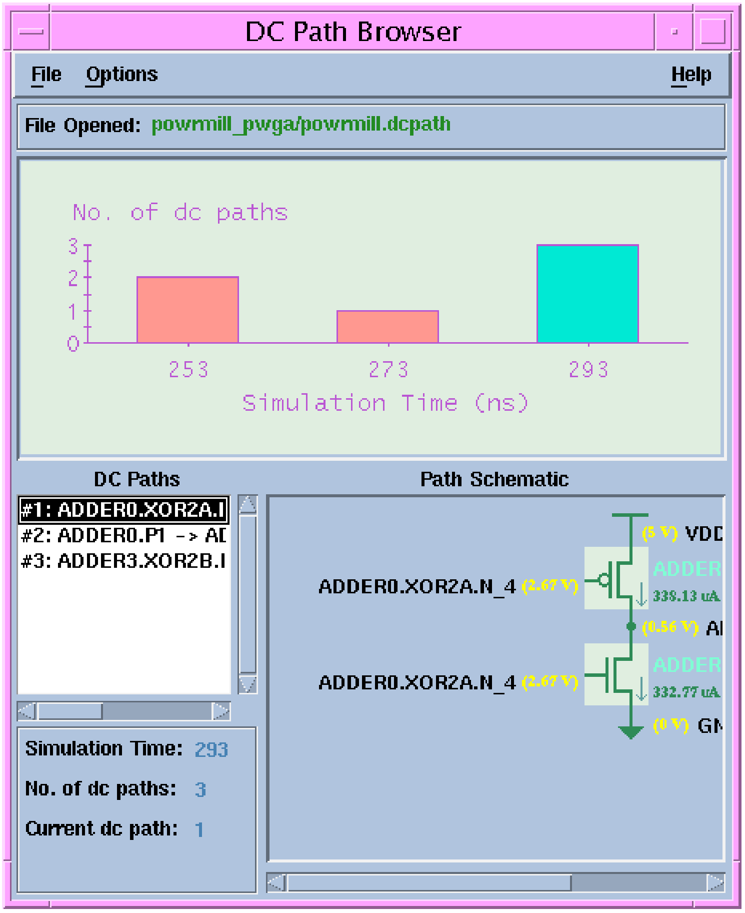


Figure 10 Power histogram browser

Select **File** → **Close Power Histogram Browser** to exit the **Power Histogram Browser**.

Viewing the DC Paths

Next, you'll view the DC paths in the circuit. Click the **DC Path Browser** button in the **Results Analysis** form. The **DC Path Browser** appears (see Figure 11).



File Opened: powrmill_pwga/powrmill.dcpath

#1: ADDER0.XOR2A.I
#2: ADDER0.P1 -> AD
#3: ADDER3.XOR2B.I

Simulation Time: 293
No. of dc paths: 3
Current dc path: 1

Figure 11 DC Path Browser

The top portion of the **DC Path Browser** graphs the number of DC paths that occur at particular time blocks. To view the paths that occur at a time block, click on a time block using your left mouse button.

Notice that a list of DC paths is displayed in the DC Paths text field located at the bottom left corner of the window. A schematic of the selected path appears in the display area located at the bottom right corner of the window.

To close the **DC Path Browser**, select **File** → **Close DC Path Browser**.

Viewing the Power Diagnostics

1. Click the **Power Diagnostics** button in the **Results Analysis** form.

   The **Power Diagnostics Browser** appears (see Figure 12).

2. To display a particular set of errors, you'll first need to select an error type in the form.

   For this design, select the following error categories:

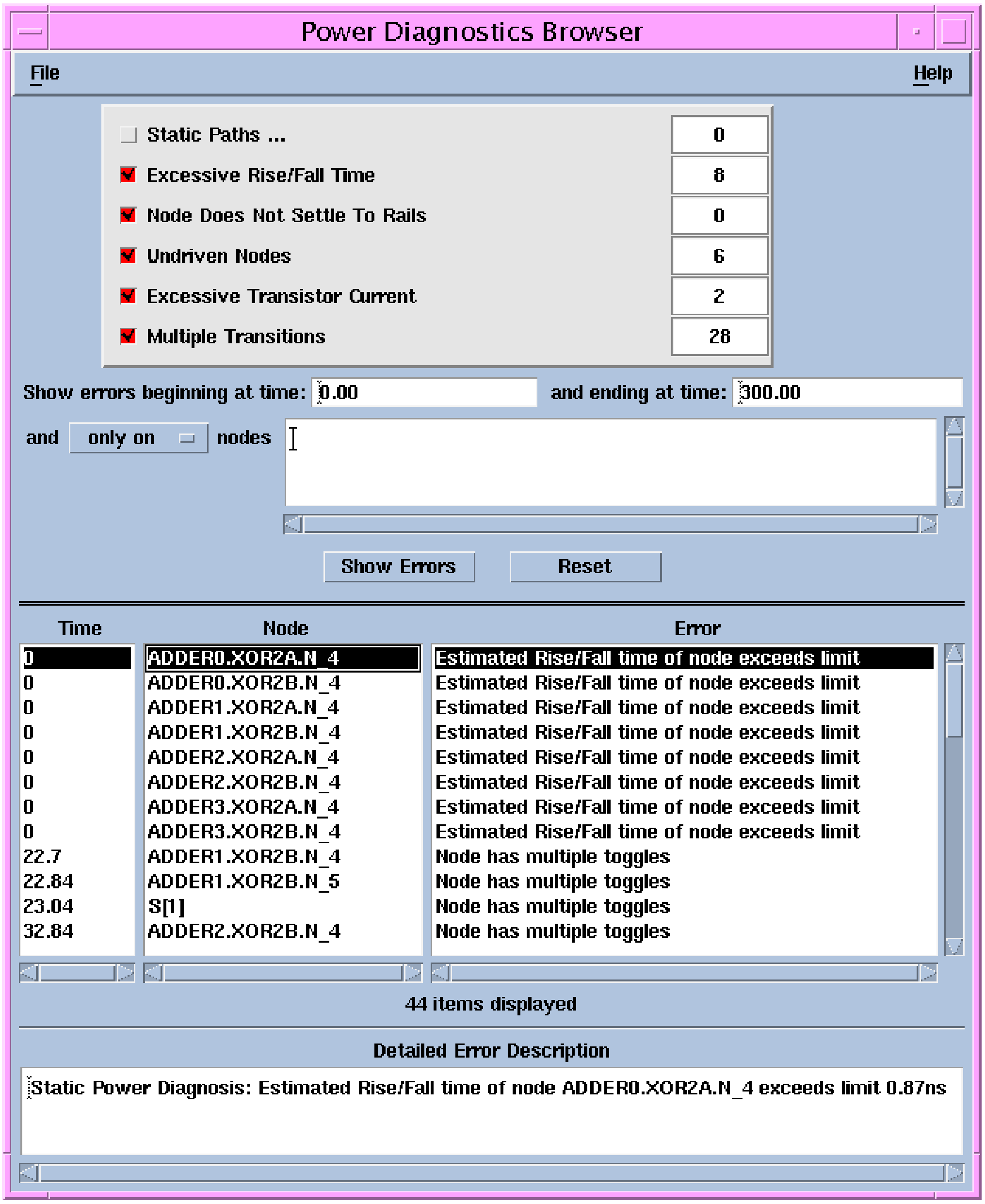
   - Excessive Rise/Fall Time
   - Undriven Nodes
   - Excessive Transistor Current
   - Multiple Transitions

3. Click the **Show Errors** button to display the errors you selected.

4. To view specific nodes, you can copy and paste node names from the list at the bottom of the window. Use the left mouse button to select the `ADDER0.XOR2A.N_4` node.

5. Next, use the middle mouse button to drag the node name to the text field above the **Show Errors** button. Click the **Show Errors** button to list only the errors that occurred on that node.

6. To exit the **Power Diagnostics Browser**, select:
   File → **Close Power Diagnostics Browser**.



☐ Static Paths ...	0
☑ Excessive Rise/Fall Time	8
☑ Node Does Not Settle To Rails	0
☑ Undriven Nodes	6
☑ Excessive Transistor Current	2
☑ Multiple Transitions	28

Show errors beginning at time: 0.00 and ending at time: 300.00

and only on nodes

Time	Node	Error
0	ADDER0.XOR2A.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER0.XOR2B.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER1.XOR2A.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER1.XOR2B.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER2.XOR2A.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER2.XOR2B.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER3.XOR2A.N_4	Estimated Rise/Fall time of node exceeds limit
0	ADDER3.XOR2B.N_4	Estimated Rise/Fall time of node exceeds limit
22.7	ADDER1.XOR2B.N_4	Node has multiple toggles
22.84	ADDER1.XOR2B.N_5	Node has multiple toggles
23.04	S[1]	Node has multiple toggles
32.84	ADDER2.XOR2B.N_4	Node has multiple toggles

44 items displayed

Static Power Diagnosis: Estimated Rise/Fall time of node ADDER0.XOR2A.N_4 exceeds limit 0.87ns

Figure 12 Power Diagnostics Browser

Displaying Waveforms for Power and Ground Nodes

Use the following procedure to display waveforms for power and ground nodes.

1 Select the waveform display tool: **Simwave** or **turbowave** from the **Options** menu in the main window.

2 Select the name of the waveform display tool from the **Result Analysis** menu.

  The waveform tool is started and the appropriate output file is loaded.

3 From the waveform tool, select the signals you want to view.

Appendix A

Sample Power Reports

Sample Block Power Report

```
Block: total
    Number of nodes in block        :   3953
    Number of elements in block     :   9752
    Number of block supply nodes    :   11
    Number of block ground nodes    :   5
    Number of block biput nodes     :   17
    Number of block input nodes     :   3
    Number of block output nodes    :   0
    Number of block stages          :   1597
    Number of block partial stages  :   0

    Average supply current          :   -45144.660642 uA
    RMS supply current              :   64875.016572 uA

    Average ground current          :   45127.965926 uA
    RMS ground current              :  64878.957946 uA

    Average input current           :   0.004319 uA
    RMS input current               :   46.694417 uA

    Average output current          :   0.000000 uA
    RMS output current              :   0.000000 uA

    Average biput current           :   16.694782 uA
    RMS biput current               :   293.473489 uA

    Average capacitive current      :   -39057.934468 uA
    RMS capacitive current          :   56567.265346 uA

    Average wasted current          :   -6115.921641 uA
```

```
RMS wasted current              :   9764.697468 uA

Wasted current percentage       :   13.538631%

Average block power             :   179937.712787 uW
RMS block power                 :   259344.959596 uW

Supply node currents:
      Node: vdq
          Average current       :   -1292.744780 uA
          RMS current           :   2689.379684 uA
      Node: vext
          Average current       :   -13.113817 uA
          RMS current           :   60.555783 uA
      Node: vr2k
          Average current       :   -1.306000 uA
          RMS current           :   1.306000 uA
      Node: vbprst
          Average current       :   0.000000 uA
          RMS current           :   0.000000 uA
      Node: vrad
          Average current       :   -0.096472 uA
          RMS current           :   125.248341 uA
      Node: vref
          Average current       :   0.000000 uA
          RMS current           :   0.000000 uA
      Node: vccex
          Average current       :   -3254.654260 uA
          RMS current           :  6735.633010 uA
      Node: vbbex
          Average current       : -46.280923 uA
          RMS current           :  65.944698 uA
      Node: vplex
          Average current       : -48.694494 uA
          RMS current           :  65.990722 uA
      Node: vblex
          Average current       : -22.313620 uA
          RMS current           :  35.782504 uA
      Node: vdd
          Average current       : -40465.456275 uA
          RMS current           :  62012.343945 uA

Ground node currents:
      Node: vwpb
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
      Node: vbfe
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
      Node: vnbl
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
      Node: vsc
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
      Node: gnd
          Average current       :  45127.965926 uA
          RMS current           :  64878.957946 uA

Input node currents:
      Node: vbwe
          Average current       :  0.001269 uA
          RMS current           :  0.014343 uA
      Node: vbcas
          Average current       :  0.001368 uA
          RMS current           :  23.447023 uA
      Node: vbras
          Average current       :  0.001682 uA
          RMS current           :  30.281411 uA

Biput node currents:
    Node: vdin
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
    Node: vdq0
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
    Node: vdq1
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
    Node: vdq2
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
    Node: vdq3
          Average current       :  0.000000 uA
          RMS current           :  0.000000 uA
    Node: vain1
          Average current       :  0.095737 uA
          RMS current           :  44.265948 uA
    Node: vain0
          Average current       :  3.260720 uA
          RMS current           :  38.813769 uA
    Node: vain2
          Average current       :  3.260651 uA
          RMS current           :  38.813809 uA
    Node: vain3
          Average current       :  0.095980 uA
          RMS current           :  44.266041 uA
    Node: vain4
          Average current       :  3.260651 uA
          RMS current           :  38.813809 uA
    Node: vain5
          Average current       :  3.260651 uA
          RMS current           :  38.813809 uA
    Node: vain6
          Average current       :  0.095980 uA
          RMS current           :  44.266041 uA
    Node: vain7
          Average current       :  0.095979 uA
          RMS current           :  44.266041 uA
    Node: vain8
          Average current       :  0.001632 uA
          RMS current           :  1.129229 uA
    Node: vain9
          Average current       :  3.263539 uA
          RMS current           :  39.380620 uA
    Node: vain10
          Average current       :  0.001632 uA
          RMS current           :  1.129229 uA
    Node: vain11
          Average current       :  0.001632 uA
          RMS current           :  1.129229 uA
```

Sample .power File

```
;! power_file_format 5.4
; -------------------------------------------------------
;|                                                       |
;|                 PowerMill Version 5.4                 |
;|                  SN: P081899-SunOS_5                  |
;| Copyright (c) 1999 Synopsys Inc., All Rights Reserved. |
;|                                                       |
; -------------------------------------------------------
;
;


*********************************************************************************
                          RUN TIME POWER BUDGET VIOLATIONS
*********************************************************************************

BLOCK top: DRAWS EXCESSIVE POWER FROM 10.8 TO 10.9 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 20.9 TO 21 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 20.8 TO 21.1 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 21.4 TO 21.5 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 21.4 TO 21.5 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 21.6 TO 21.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 21.6 TO 21.9 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 22.2 TO 22.5 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 22.2 TO 22.5 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 50.4 TO 50.5 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 50.4 TO 50.5 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 61.4 TO 61.5 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 61.4 TO 61.5 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 61.6 TO 62.4 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 61.6 TO 62.4 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 101.6 TO 101.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 101.6 TO 101.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 102.2 TO 102.3 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 102.2 TO 102.3 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 141.6 TO 141.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 141.6 TO 141.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 142.2 TO 142.4 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 142.2 TO 142.4 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 170.8 TO 170.9 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 181.4 TO 181.5 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 181.4 TO 181.5 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 181.6 TO 181.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 181.6 TO 181.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 182.2 TO 182.4 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 182.2 TO 182.4 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 221.6 TO 221.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 221.6 TO 221.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 222.2 TO 222.3 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 222.2 TO 222.3 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 261.6 TO 261.7 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 261.6 TO 261.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 262.2 TO 262.4 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 262.2 TO 262.4 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 301.6 TO 301.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 301.6 TO 301.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 302.2 TO 302.3 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 302.2 TO 302.3 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 341.6 TO 341.8 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 341.6 TO 341.8 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 342.2 TO 342.4 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 342.2 TO 342.4 ns.
BLOCK top: DRAWS EXCESSIVE SUPPLY CURRENT FROM 381.6 TO 381.7 ns.
BLOCK top: DRAWS EXCESSIVE POWER FROM 381.6 TO 381.7 ns.

*********************************************************************************
                          POWER BUDGET VIOLATION SUMMARY
*********************************************************************************

BLOCK top: MAXIMUM INSTANTANEOUS SUPPLY CURRENT OF -157249 uA EXCEEDS BUDGET OF 30000 uA.
BLOCK top: AVERAGE SUPPLY CURRENT OF -1537.84 uA EXCEEDS BUDGET OF 1000 uA.
BLOCK top: MAXIMUM INSTANTANEOUS POWER OF 566098 uW EXCEEDS BUDGET OF 100000 uW.
BLOCK top: AVERAGE POWER OF 5536.21 uW EXCEEDS BUDGET OF 3000 uW.

*********************************************************************************
                          BLOCK HIERARCHICAL POWER ANALYSIS
*********************************************************************************



BLOCK top: AVERAGE SUPPLY CURRENT.

    LEVEL            CURRENT     PERCENT OF    PERCENT OF    CHILD BLOCK NAME
                      (uA)         PARENT         TOP
----------------------------------------------------------------------------------
-------------
*---------   0       -1537.8       100.00        100.00       top
-*--------   1       -1537.8       100.00        100.00       top_xsr34x22
--*-------   2       -359.83        23.40         23.40       top_xsr34x22.x1161
--*-------   2       -322.71        20.98         20.98       top_xsr34x22.x1164
--*-------   2       -214.54        13.95         13.95       top_xsr34x22.x1166
--*-------   2       -207.33        13.48         13.48       top_xsr34x22.x1158
--*-------   2       -192.92        12.55         12.55       top_xsr34x22.x1162
--*-------   2       -81.777         5.32          5.32       top_xsr34x22.x1163
--*-------   2       -57.569         3.74          3.74       top_xsr34x22.x1157
--*-------   2       -52.966         3.44          3.44       top_xsr34x22.x1160
--*-------   2       -42.359         2.75          2.75       top_xsr34x22.x1169
--*-------   2       -5.8502         0.38          0.38       top_xsr34x22.x1167
--*-------   2     0.0086932        -0.00         -0.00       top_xsr34x22.x1170
--*-------   2             0        -0.00         -0.00       top_xsr34x22.x1165
--*-------   2             0        -0.00         -0.00       top_xsr34x22.x1168
--*-------   2             0        -0.00         -0.00       top_xsr34x22.x1159


BLOCK top: RMS SUPPLY CURRENT.

    LEVEL            CURRENT     PERCENT OF    PERCENT OF    CHILD BLOCK NAME
                      (uA)         PARENT         TOP
----------------------------------------------------------------------------------
-------------
*---------   0        6959.2       100.00        100.00       top
-*--------   1        6959.2       100.00        100.00       top_xsr34x22
--*-------   2        3717.1        53.41         53.41       top_xsr34x22.x1161
--*-------   2        2966.7        42.63         42.63       top_xsr34x22.x1164
--*-------   2        1785.1        25.65         25.65       top_xsr34x22.x1166
--*-------   2        1584.1        22.76         22.76       top_xsr34x22.x1162
--*-------   2        1332.5        19.15         19.15       top_xsr34x22.x1158
--*-------   2        1202.1        17.27         17.27       top_xsr34x22.x1163
--*-------   2        647.76         9.31          9.31       top_xsr34x22.x1157
--*-------   2        408.87         5.88          5.88       top_xsr34x22.x1160
--*-------   2        316.96         4.55          4.55       top_xsr34x22.x1169
```

```
--*-------   2        115.03          1.65          1.65       top_xsr34x22.x1167
--*-------   2         2.588          0.04          0.04       top_xsr34x22.x1170
--*-------   2             0          0.00          0.00       top_xsr34x22.x1165
--*-------   2             0          0.00          0.00       top_xsr34x22.x1168
--*-------   2             0          0.00          0.00       top_xsr34x22.x1159


BLOCK top: AVERAGE POWER.

    LEVEL              POWER      PERCENT OF    PERCENT OF    CHILD BLOCK NAME
                       (uW)         PARENT         TOP
-----------------------------------------------------------------------------------
-------------
*---------   0        5536.2        100.00        100.00       top
-*--------   1        5536.2        100.00        100.00       top_xsr34x22
--*-------   2        1467.9         26.51         26.51       top_xsr34x22.x1161
--*-------   2        746.39         13.48         13.48       top_xsr34x22.x1158
--*-------   2        722.92         13.06         13.06       top_xsr34x22.x1166
--*-------   2        694.52         12.55         12.55       top_xsr34x22.x1162
--*-------   2        639.25         11.55         11.55       top_xsr34x22.x1164
--*-------   2        430.29          7.77          7.77       top_xsr34x22.x1163
--*-------   2        258.27          4.67          4.67       top_xsr34x22.x1167
--*-------   2        207.25          3.74          3.74       top_xsr34x22.x1157
--*-------   2        190.68          3.44          3.44       top_xsr34x22.x1160
--*-------   2        152.49          2.75          2.75       top_xsr34x22.x1169
--*-------   2        49.414          0.89          0.89       top_xsr34x22.x1165
--*-------   2     -0.031295         -0.00         -0.00       top_xsr34x22.x1170
--*-------   2             0          0.00          0.00       top_xsr34x22.x1168
--*-------   2             0          0.00          0.00       top_xsr34x22.x1159


BLOCK top: RMS POWER.

    LEVEL              POWER      PERCENT OF    PERCENT OF    CHILD BLOCK NAME
                       (uW)         PARENT         TOP
-----------------------------------------------------------------------------------
-------------
*---------   0         25053        100.00        100.00       top
-*--------   1         25053        100.00        100.00       top_xsr34x22
--*-------   2         14856         59.30         59.30       top_xsr34x22.x1161
--*-------   2        8329.2         33.25         33.25       top_xsr34x22.x1164
--*-------   2          6077         24.26         24.26       top_xsr34x22.x1166
--*-------   2        5702.9         22.76         22.76       top_xsr34x22.x1162
--*-------   2        4995.3         19.94         19.94       top_xsr34x22.x1163
--*-------   2        4796.9         19.15         19.15       top_xsr34x22.x1158
--*-------   2        2787.9         11.13         11.13       top_xsr34x22.x1167
--*-------   2        2331.9          9.31          9.31       top_xsr34x22.x1157
--*-------   2        1471.9          5.88          5.88       top_xsr34x22.x1160
--*-------   2        1141.1          4.55          4.55       top_xsr34x22.x1169
--*-------   2        602.31          2.40          2.40       top_xsr34x22.x1165
--*-------   2        9.3168          0.04          0.04       top_xsr34x22.x1170
--*-------   2             0          0.00          0.00       top_xsr34x22.x1168
--*-------   2             0          0.00          0.00       top_xsr34x22.x1159
```

Appendix B

Using Shared Memory

Many 32-bit machines limit the virtual memory space to 2GB. If you need to simulate a very large circuit that requires more than 2GB of memory, you need to set the **EPIC_USE_SHM** environment variable before running PowerMill.

EXAMPLE:

```
setenv EPIC_USE_SHM
```

This setting causes the simulator to allocate memory from a region that is shared by all the processes running on your machine. This feature is only available on HP-UX and Solaris platforms.

Combined Index

Index