

RandomSpice User Guide

Version T-2022.03, March 2022



Copyright and Proprietary Information Notice

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

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

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

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

www.synopsys.com

Contents

Conventions	5
Customer Support	5
<hr/>	
1. Introduction to the RandomSpice Tool	7
RandomSpice Functionality	7
TCAD Sentaurus Tutorial: Simulation Projects	7
Starting RandomSpice	8
Overriding Arguments in the Input File	9
Basic Simulation Procedure	11
Quick Start Example	13
Simulation Modes	16
Interconnect Variability	16
Parallel Processing on a Workstation	18
<hr/>	
2. Input Files	19
Structure of Input Files	19
Circuit Section	20
Simulation Section	21
Output Section	22
Models Section	23
Variability Section	24
Advanced Section	24
Summary of Input File Sections	25
Autogenerating Input Files	26
Preparing a Netlist	28
Subcircuit Randomization	29
Plug-in Postprocessing Modules	30
Python Library Modules	31

Contents

3. Compact Modeling Strategies	32
Subcircuit Device Models.....	32
Response Surface Models.....	33
Statistical Compact Models	34
Gaussian Parameter Generation.....	35
Principal Component Analysis	36
ModelGen Approach	37
References.....	38

4. Generating RandomSpice Libraries	39
Library Builder	39
Library Builder Example.....	39

Glossary	42
-----------------------	----

Index	43
--------------------	----

About This Guide

This document describes the functionality of the Synopsys® RandomSpice tool.

For additional information, see:

- The TCAD Sentaurus™ release notes, available on the Synopsys SolvNetPlus support site (see [Accessing SolvNetPlus](#))
- Documentation available on the SolvNetPlus support site

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field or an option.
<code>Courier font</code>	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.
<i>Italicized text</i>	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
Menu > Command	Indicates a menu command, for example, File > New (from the File menu, choose New).

Customer Support

Customer support is available through the Synopsys SolvNetPlus support site and by contacting the Synopsys support center.

Accessing SolvNetPlus

The SolvNetPlus support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you

access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNetPlus site:

1. Go to <https://solvnetplus.synopsys.com>.
2. Enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys [Global Support Centers](#) site on www.synopsys.com. There you can find email addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNetPlus site or the Synopsys Global Support Centers site and open a case (Synopsys user name and password required).

Contacting Your Local TCAD Support Team Directly

Send an email message to:

- support-tcad-us@synopsys.com from within North America and South America
- support-tcad-eu@synopsys.com from within Europe
- support-tcad-ap@synopsys.com from within Asia Pacific (China, Taiwan, Singapore, Malaysia, India, Australia)
- support-tcad-kr@synopsys.com from Korea
- support-tcad-jp@synopsys.com from Japan

1

Introduction to the RandomSpice Tool

This chapter introduces the RandomSpice tool.

RandomSpice Functionality

RandomSpice is a Monte Carlo circuit simulator, which is part of the TCAD to SPICE workflow. It performs large-scale simulations of statistical and process variability and statistical analysis of the simulation results. It supports advanced statistical compact model extraction and parameter generation strategies linked to the statistical SPICE model extraction tool Mystic.

RandomSpice efficiently harnesses large-scale cluster computing technology. It provides the functionality necessary for accurate statistical circuit and standard cell characterization and supports performance–power–yield (PPY) analysis.


RandomSpice can use the Synopsys PrimeSim™ HSPICE® tool as its backend SPICE simulator.

This document describes how to use RandomSpice to perform variability-enhanced simulations of nanoscale circuit designs.

TCAD Sentaurus Tutorial: Simulation Projects

The TCAD Sentaurus™ Tutorial provides various projects demonstrating the capabilities of RandomSpice.

To access the TCAD Sentaurus Tutorial:

1. Open Sentaurus Workbench by entering the following on the command line: `swb`
2. From the menu bar of Sentaurus Workbench, choose **Help > Training** or click  on the toolbar.

Alternatively, to access the TCAD Sentaurus Tutorial:

1. Go to the `$STROOT/tcad/current/Sentaurus_Training` directory.

The `STROOT` environment variable indicates where the Synopsys TCAD distribution has been installed.

2. Open the `index.html` file in your browser.

Starting RandomSpice

RandomSpice is a command-line program, which is invoked using the command:

```
$ RandomSpice2 <input_file> [<options>]
```

[Table 1](#) lists the command-line options available for RandomSpice. They always override the arguments given in the input file, as indicated in the Input file argument column, where the corresponding input file argument is denoted in the form *Section::Argument* (see [Chapter 2 on page 19](#)).

These command-line options can be used, for example, to change the number of circuits or the SPICE simulator without changing the input file.

Table 1 *Command-line options*

Option (short form followed by long form if applicable)	Description	Input file argument
<code>-c <integer></code> <code>--start-circuit <integer></code>	Sets the starting circuit number in the statistical ensemble.	<code>Circuit::startnum</code>
<code>-d <string></code> <code>--output-dir <string></code>	Specifies the path to the output directory to use.	<code>Output::dir</code>
<code>-g</code> <code>--gen-only</code>	Specifies to generate circuits only (that is, RandomSpice does not run any simulations).	<code>Simulation::enabled</code>
<code>-h</code> <code>--help</code>	Displays command-line options and usage.	–
<code>-i</code> <code>--gen-input</code>	Generates a template <code>.rs2</code> file based on the supplied arguments.	–
<code>-L <string></code> <code>--library <string></code>	Specifies the file name of the compact model library to use.	<code>Models::library</code>

Chapter 1: Introduction to the RandomSpice Tool

Starting RandomSpice

Table 1 *Command-line options (Continued)*

Option (short form followed by long form if applicable)	Description	Input file argument
-m <string> --sim-mode <string>	Sets the simulation mode to use (see Simulation Modes on page 16).	Simulation::mode
-n <integer>	Sets the number of randomized circuits to simulate.	Circuit::number
-o <string> --prefix <string>	Specifies the prefix to use for the names of output files.	Output::prefix
-q --query-lib	Queries the compact model library to determine which device keywords are available.	-
-r <integer>	Sets the seed for the random number generator.	Circuit::seed
-s <string> --spice <string>	Specifies the SPICE simulator to use.	Simulation::spice
--spice-args <string>	Specifies additional arguments to be passed to the SPICE simulator.	Simulation::spiceargs
-t <string> --template <string>	Specifies the file name of the template netlist to use.	Circuit::netlist
-v --version	Displays the RandomSpice version.	-

Overriding Arguments in the Input File

As an illustration, for the example in [Quick Start Example on page 13](#), you can override the number of circuit simulations in the input file using the `-n` command-line option, so that 50 simulations are performed instead of the 10 specified in the example:

```
$ RandomSpice2 inv.rs2 -n 50
=====
TCAD to SPICE - RandomSpice

Version T-2022.03 for linux64

Copyright (c) 2010-2022 Synopsys, Inc.
This software and all associated documentation are proprietary to
```

Chapter 1: Introduction to the RandomSpice Tool

Starting RandomSpice

```
Synopsys, Inc. and may only be used pursuant to the terms and conditions
of a written license agreement with Synopsys, Inc. All other use,
reproduction, modification, or distribution of this software
or the associated documentation is strictly prohibited.
=====
Loading inputs:
  RandomSpice input file: /home/username/Desktop/
                          RandomSpice_Tutorial/inv.rs2
  Template netlist:      /home/username/Desktop/
                          RandomSpice_Tutorial/inv.net
  Library file:          $INSTALL_DIR/Models/GSS-25nm-HP-Metal/
                          GSS-25nm-HP-Metal-Models.rsl
  Post-processor:        Disabled
Loading complete

Simulation Parameters:
  Model Library:         GSS 25nm HP Metal - LUT v1.0.0
  SV Enabled:            True
  PV Enabled:            False
  Initial Seed:          1000
  SPICE Backend:         ngspice
  Sim Mode:              Normal
  # of Circuits:         50
  Starting Circuit:      1

Netlist generation:
[=====] 100%

Simulating circuits:
[=====] 100%

Output data:
Netlists saved to /home/username/Desktop/RandomSpice_Tutorial
Simulation data saved to /home/username/Desktop/RandomSpice_Tutorial

Total run time: 0hr 0min 11s
```

Note:

Running many simulations can rapidly produce a lot of output data. It is suggested that you use an alternative storage format, such as GZIP or BZIP2. For the options available for file storage, see [Output Section on page 22](#) (backend argument). Large simulation samples might also require large amounts of memory and, for this reason, RandomSpice offers different simulation modes to manage this (see [Simulation Modes on page 16](#)).

Basic Simulation Procedure

Note:

Before running RandomSpice simulations, ensure that the SPICE simulator is available and properly configured within the environment in which RandomSpice is running.

Requirements for basic RandomSpice simulations include:

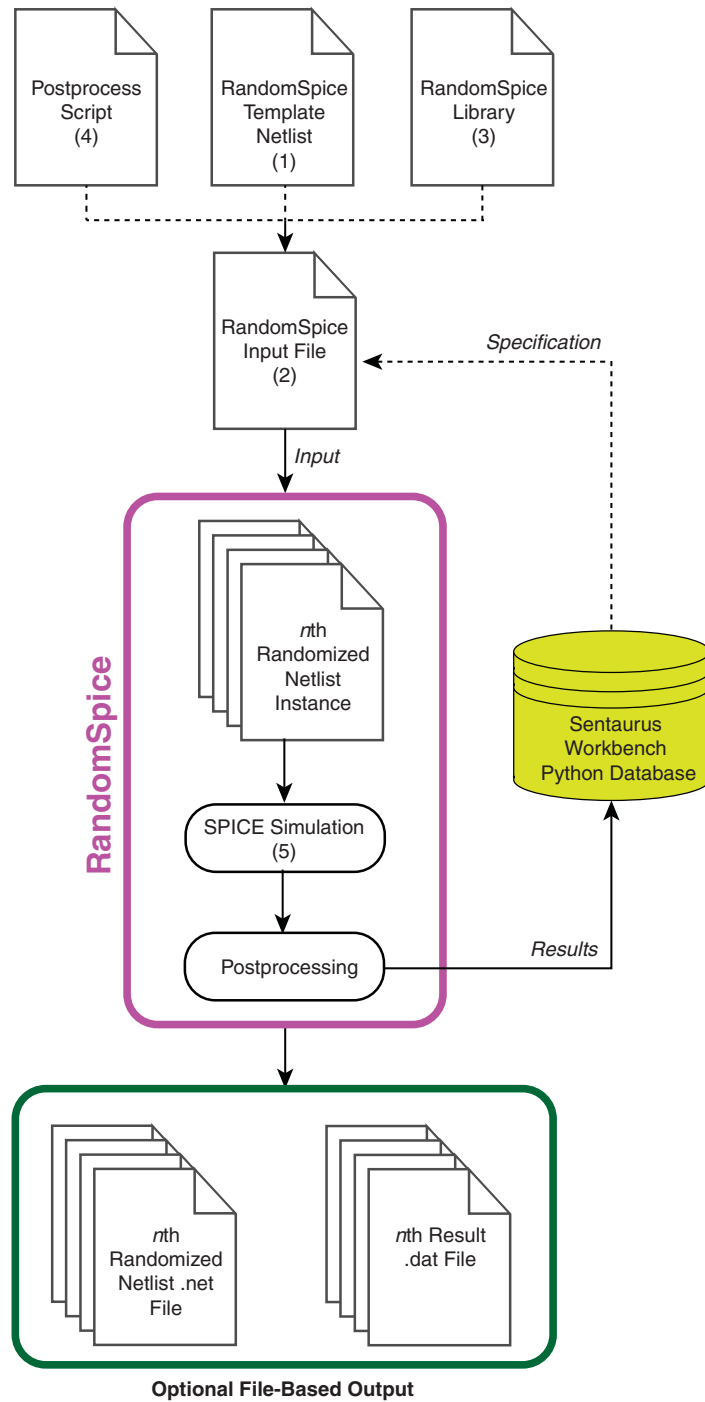
- Netlist or circuit for simulation, extracted from a schematic design or layout, or written manually, including required options, stimulus, and measurements. Changes required to standard SPICE netlists are outlined in [Preparing a Netlist on page 28](#). This required input is denoted by (1) in [Figure 1 on page 12](#).
- Input file containing simulation setup and settings (see [Structure of Input Files on page 19](#)). This required input is denoted by (2) in [Figure 1](#).
- Model library with generators, as specified in the input file. This required input is denoted by (3) in [Figure 1](#).
- Processor to extract data from the simulation output (see [Plug-in Postprocessing Modules on page 30](#)). This optional input is denoted by (4) in [Figure 1](#).
- SPICE simulator to perform the individual simulations. This required input is denoted by (5) in [Figure 1](#).

Before RandomSpice simulations are performed, you must simulate the nominal circuit with the SPICE simulator to verify that the circuit performs as expected. You must also perform an initial run with a small number of circuit instances first, where generated netlists are saved (an available option in the input file) and analyzed for troubleshooting and debugging purposes.

Chapter 1: Introduction to the RandomSpice Tool

Basic Simulation Procedure

Figure 1 *Inputs, outputs, and general RandomSpice workflow: solid arrows indicate the workflow and dashed lines indicate where specifications are input into RandomSpice, mainly through the input file (2)*



Quick Start Example

This example demonstrates how to work with RandomSpice, using a simple CMOS inverter circuit. The input files for this example are in the following directory:

```
$INSTALL_DIR/RandomSpice2/Examples/QuickStart
```

The template SPICE netlist is provided in `inv.net`, which is reproduced here. This is a regular SPICE netlist, with one important difference: the model names used for the MOSFETs in the circuit are replaced with keywords that instruct RandomSpice to create a model for this MOSFET (see [Preparing a Netlist on page 28](#)).

Different keywords might be available in different libraries, for example, to allow different variability sources to be applied. You can specify the `-q` command-line option to query a compact model library to find out which keywords are available. For example:

```
$ RandomSpice2 Test.rs2 -q
=====
TCAD to SPICE - RandomSpice

Version T-2022.03 for linux64

Copyright (c) 2010-2022 Synopsys, Inc.
This software and all associated documentation are proprietary to
Synopsys, Inc. and may only be used pursuant to the terms and conditions
of a written license agreement with Synopsys, Inc. All other use,
reproduction, modification, or distribution of this software
or the associated documentation is strictly prohibited.
=====

Querying library GSS-25nm-HP-Metal-Models.rsl

Library Name:      GSS 25nm HP Metal Gate Library - LUT Models
Supported Keywords: UNIF:N, UNIF:P, ATOM:N, ATOM:P

Total run time: 0hr 0min 4s
```

Note that an input file is still required, which specifies the library file name.

The examples included in this document use the GSS-25nm-Bulk-MOSFET library, which contains the device keywords `ATOM:N` (denoting an n-channel device containing variability) and `ATOM:P` (denoting a variable p-channel device).

```
* CMOS Inverter

M1 Y A 0 0 ATOM:N L=25n W=25n
M2 Y A VDD VDD ATOM:P L=25n W=50n

VDD VDD 0 1
V1 A 0 PULSE(0 1 0 10p 10p 1n 2n)
```

Chapter 1: Introduction to the RandomSpice Tool

Basic Simulation Procedure

```
.TRAN 10p 10n
.PRINT TRAN V(A) V(Y)

.OPTIONS NOPAGE=1 INGOLD=2
.END
```

The second required input file configures RandomSpice for the simulation that you want to run. The input file used for this example is `inv.rs2` in the `QuickStart` directory, which is reproduced here. All of the arguments are explained in [Chapter 2 on page 19](#), but the important arguments to note are:

- `netlist` provides RandomSpice with the file name of the template SPICE netlist.
- `number` is the number of simulations to run.
- `library` provides RandomSpice with the name of the compact model library to use.
- `spice` provides RandomSpice with the SPICE simulator to use.

The other arguments in this input file configure output file names, formats, and so on.

```
[Circuit]
netlist = inv.net
number = 10
seed = 1000

[Simulation]
spice = ngspice

[Output]
dir = ./
prefix = inv
savenets = True
backend = file

[Models]
library = $INSTALL_DIR/Models/GSS-25nm-HP-Metal/
GSS-25nm-HP-Metal-Models.rsl
```

You should copy these two files from the `QuickStart` directory to your own workspace (as shown here), and then invoke RandomSpice on the command line using the following command:

```
$ mkdir RandomSpice_Tutorial
$ cd RandomSpice_Tutorial
$ cp -v $INSTALL_DIR/RandomSpice2/Examples/QuickStart/inv.* ./
'$INSTALL_DIR/RandomSpice2/Examples/QuickStart/inv.net' -> './
inv.net' '$INSTALL_DIR/RandomSpice2/Examples/QuickStart/inv.rs2' -> './
inv.rs2'
$ RandomSpice2 inv.rs2
=====
TCAD to SPICE - RandomSpice
```

Chapter 1: Introduction to the RandomSpice Tool

Basic Simulation Procedure

Version T-2022.03 for linux64

Copyright (c) 2010-2022 Synopsys, Inc.

This software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of this software or the associated documentation is strictly prohibited.

=====

Loading inputs:

RandomSpice input file: /home/username/Desktop/
RandomSpice_Tutorial/inv.rs2
Template netlist: /home/username/Desktop/
RandomSpice_Tutorial/inv.net
Library file: \$INSTALL_DIR/Models/GSS-25nm-HP-Metal/
GSS-25nm-HP-Metal-Models.rsl
Post-processor: Disabled

Loading complete

Simulation Parameters:

Model Library: GSS 25nm HP Metal - LUT v1.0.0
SV Enabled: True
PV Enabled: False
Initial Seed: 1000
SPICE Backend: ngspice
Sim Mode: Normal
of Circuits: 10
Starting Circuit: 1

Netlist generation:

[=====] 100%

Simulating circuits:

[=====] 100%

Output data:

Netlists saved to /home/username/Desktop/RandomSpice_Tutorial
Simulation data saved to /home/username/Desktop/RandomSpice_Tutorial

Total run time: 0hr 0min 9s

This produces 10 randomized instances of the template inverter, with the output netlists saved as inv-1.net to inv-10.net, and a corresponding .dat file for each circuit instance that contains the screen output captured from the SPICE simulator. Using this output data, you can perform further postprocessing manually or develop a plug-in postprocessor for RandomSpice.

Simulation Modes

RandomSpice provides different simulation modes, which account for differences between the computational environments in which RandomSpice can run. These modes provide trade-offs between simulation runtime and memory footprint:

- In *normal* mode (the default), all circuits for an ensemble are generated, then simulations are performed, with postprocessing for each circuit performed as soon as the simulation finishes. This is the best compromise between simulation time and memory footprint.
- In *batch* mode, all circuits for an ensemble are generated, then all simulations are performed, followed by all postprocessing. This is the fastest mode, but it can require large amounts of memory, depending on the circuit size. This mode is most useful when circuit simulations generate only a small amount of data.
- In *sequential* mode, each circuit is generated, simulated, and postprocessed in turn. This mode is slower, but it can significantly reduce the memory footprint and is most useful when simulations generate a large amount of output data.

You can specify the simulation mode using the `-m` command-line option or the `Simulation` section of the input file (see [Simulation Section on page 21](#)).

By default, spooling is activated for each simulation mode, meaning that when the amount of stored data reaches a certain limit, it is stored temporarily to disk in a spool file until it is needed again. This is a transparent process where all spool files are cleaned up, and the output data is returned to you at the end of the simulation. By default, RandomSpice is configured to limit the memory usage to approximately 2 GB for simulation data, although there is additional overhead for RandomSpice itself and the SPICE simulator. For more information, see [Advanced Section on page 24](#).

Interconnect Variability

RandomSpice can include variations in interconnect components (resistors, capacitors, and inductors). This works very similarly to introducing variations in MOSFET components, and this section presents an example illustrating how you can use this feature.

In conjunction with the Library Builder utility, you can insert variations coming either from RC extraction or as specified by a parametric distribution. This includes access to the Gaussian, principal component analysis (PCA), and ModelGen generators.

To introduce variations in interconnect components, replace the component value with the keyword specified by the library, for example, `ATOM:C`, `ATOM:R`, or `ATOM:L`. Since correlated variations can be applied to different components, you must specify to which component in an interconnect model the variations refer. For example, in a simplistic model with only one resistor and one capacitor, these would be referred to as `ATOM:R:1` and `ATOM:C:1`.

Chapter 1: Introduction to the RandomSpice Tool

Interconnect Variability

To illustrate this, consider the following netlist:

```
* Interconnect example

MN1 Y1 A 0 0 ATOM:N L=25n W=25n
MP1 Y2 A VDD VDD ATOM:P L=25n W=25n

R1 Y1 Y2 1
C1 Y2 0 1e-15

.END
```

This would be altered as follows:

```
* Interconnect example

MN1 Y1 A 0 0 ATOM:N L=25n W=25n
MP1 Y2 A VDD VDD ATOM:P L=25n W=25n

R1 Y1 Y2 ATOM:R:1
C1 Y2 0 ATOM:C:1

.END
```

Then after randomization, a particular instance might look like:

```
* Interconnect example

MN1 Y1 A 0 0 ATOM:N L=25n W=25n
MP1 Y2 A VDD VDD ATOM:P L=25n W=25n

R1 Y1 Y2 1.231
C1 Y2 0 0.964e-15

.END
```

No further modifications to the netlist are necessary other than the keyword substitution. This can be extended to arbitrarily complex interconnect models with multiple correlated components. You can also use this approach to introduce variations in any equivalent circuit that can be represented entirely as passive (that is, R, L, and C) components.

Note:

The library used with RandomSpice must contain information about passive component variation for this to work correctly. You can check this with the `-q` command-line option. If there is no passive variation information in the library, the RLC components remain as is.

Parallel Processing on a Workstation

Although RandomSpice does not support native parallel processing, you can run multiple instances on the same host, allowing you to take advantage of multiprocessors.

These simulations can be set up in the same way as other simulations. The only required modification is to use either the `-c` command-line option or the `startnum` argument of the `Circuit` section in the input file, which splits the job into chunks for each processor to run. Using `-c` or `startnum` ensures that the netlists use the correct random seed.

For example, a simulation of 10000 circuits can be split into 8 jobs of 1250 circuits. Multiple instances of RandomSpice can then be invoked as follows:

```
$ RandomSpice2 mysim.rs2 -n 1250 -d Job$i -c ($i*1250)+1
```

where `$i` is the job number from 0 to 7.

2

Input Files

This chapter describes the input files required by RandomSpice.

Structure of Input Files

RandomSpice requires two files to operate:

- A SPICE netlist that is used as a template for randomization
- An input file that specifies the settings for a particular simulation

The input file is organized into different sections, prefaced by a heading, with each section having arguments (see [Table 2 on page 20](#)). For example:

```
[<section_heading1>]
<argument> = <value>
...
[<section_heading2>]
<argument> = <value>
...
```

Note:

Arguments and section headings are case insensitive. The input file is entirely unstructured, and sections can be in any order.

You can combine these two input files into one RandomSpice *hybrid* input file, which includes both RandomSpice settings and the SPICE simulation netlist. In this case, the RandomSpice settings section must start with the `.rs2` command and must end with the `.endrs2` command. For example:

```
.rs2
[<section_heading1>]
<argument> = <value>
...
[<section_heading2>]
<argument> = <value>
.endrs2
<SPICE netlist>
```

Chapter 2: Input Files

Structure of Input Files

In this hybrid input file mode, the `netlist` argument of the `Circuit` section is not required, as the netlist and input file are combined.

Table 2 Sections in input file

Section	Description
<code>Circuit</code>	Sets arguments for the template circuit, such as the number of randomized circuits to simulate (see Circuit Section).
<code>Simulation</code>	Sets arguments for SPICE simulations, such as which simulator to use (see Simulation Section on page 21).
<code>Output</code>	Sets arguments controlling how data is output from RandomSpice (see Output Section on page 22).
<code>Models</code>	Sets arguments for the compact model library used for simulations (see Models Section on page 23).
<code>Variability</code>	Sets arguments controlling different variability sources (see Variability Section on page 24).
<code>Advanced</code>	Sets advanced arguments controlling the internal operation of RandomSpice (see Advanced Section on page 24).

Circuit Section

The arguments available under the `Circuit` section are listed here.

Argument	Description	Default
<code>netlist=<string></code>	Sets the file name of the template SPICE netlist. This argument is not required if running the simulation in hybrid input file mode.	<code>./in.net</code>
<code>number=<integer></code>	Sets the number of randomized circuits to simulate.	1

Argument	Description	Default
<code>seed=<integer></code>	Sets the value used to seed the random number generator ¹ . If the default value is used, a random seed is generated based on the system time. This must not be used in a cluster or multiprocessor environment, where multiple jobs might start within the same second. The simulation will also not be reproducible unless the random seed is extracted from the simulation outputs and entered into the input file. RandomSpice uses the Mersenne Twister random number generator. This generator has been extensively tested and is considered the produce very high quality random numbers.	0
<code>startnum=<integer></code>	Sets the number of the first circuit in the statistical ensemble ¹ .	1

1. This argument specifies the base seed used by the random number generator in RandomSpice for the statistical ensemble. Each circuit is assigned a new random seed to ensure that results can be reproduced. The seed for a particular circuit is given by `seed + startnum`. Therefore, when simulating multiple ensembles, consecutive seeds must not be used, as this will result in duplicate circuits. Instead, the seeds must be separated by the ensemble size or more. See [Parallel Processing on a Workstation on page 18](#) for an example of proper usage in a cluster environment.

Simulation Section

The arguments available under the `Simulation` section are listed here.

Argument	Description	Default
<code>enabled=<string></code>	Specifies whether to run SPICE simulations. If you switch off simulations, that is, <code>enabled=False</code> , only the netlist generation stage is performed. This might be useful for debugging and testing.	True
<code>mode=<string></code>	Sets the simulation mode to use in RandomSpice. Options are: <ul style="list-style-type: none">• <code>batch</code>• <code>normal</code>• <code>sequential</code> Different simulation modes account for the particular environments in which RandomSpice might operate (see Simulation Modes on page 16).	normal

Chapter 2: Input Files

Structure of Input Files

Argument	Description	Default
<code>spice=<string></code>	Sets the SPICE simulator to use. Options are: <ul style="list-style-type: none">• <code>hspice</code>• <code>ngspice</code> For legacy reasons, RandomSpice provides limited support for the open-source SPICE simulator ngspice. To use a different SPICE simulator that might be available to you, specify it in the input file or on the command line.	<code>hspice</code>
<code>spiceargs=<string></code>	Sets additional arguments to be passed to the SPICE simulator. You can specify additional arguments that will be used when calling the SPICE simulator. For example, to use the PrimeSim HSPICE tool in server mode, you can start a PrimeSim HSPICE server with <code>hspice -C</code> before running RandomSpice, then set <code>spiceargs</code> to <code>-C</code> to instruct the PrimeSim HSPICE tool, as invoked by RandomSpice, to use the running server. This mode of use can speed up simulations where large numbers of small simulations are performed. Other arguments that can activate particular options or outputs in the SPICE simulator can also be passed this way.	–

Output Section

The arguments available under the `Output` section are listed here.

Argument	Description	Default
<code>backend=<string></code>	Sets how to store output data. Options are: <ul style="list-style-type: none">• <code>bz2, bzip2</code>: Create a <code>bzip2</code> tar file.• <code>file</code>: Create one output file for each circuit.• <code>gz, gzip</code>: Create a gzipped tar file.• <code>single</code>: Create one file containing all the output data. This option is most useful in conjunction with a postprocessing module that returns only a single number.• <code>tar</code>: Create a tar archive containing the output files.	–
<code>dir=<string></code>	Sets the output directory in which to save results.	<code>./</code>

Chapter 2: Input Files

Structure of Input Files

Argument	Description	Default
<code>prefix=<string></code>	Sets the prefix for output files. Output files are named using the <code><prefix>-<circuitnum>.[net dat]</code> convention, unless <code>backend=single</code> , in which case, the output file names are <code><prefix>.[net dat]</code> . If a file already exists with the same name, the output is renamed to include the current date and time, so as not to overwrite existing data.	<code>out</code>
<code>processor=<string></code>	Sets the file name of the Python postprocessing module to use. RandomSpice allows the creation of plug-in postprocessing modules in the Python language that can be used to process the data output by your simulation as part of the simulation workflow. See Plug-in Postprocessing Modules on page 30 for details and examples.	<code>none</code>
<code>savenets=<string></code>	Specifies whether to save generated netlists.	<code>False</code>

Models Section

The arguments available under the `Models` section are listed here.

Argument	Description	Default
<code>extrapolate=<string></code>	Specifies whether to allow the response surface model to extrapolate outside the boundaries of the input data. This argument only works with <code>rsm_method=rsm</code> .	<code>False</code>
<code>instance_params_<KEYWORD>=<Python_dict></code>	Specifies a dictionary of instance parameters to be added to every device matching this RandomSpice <code><KEYWORD></code> .	<code>{}</code>
<code>library=<string></code>	Sets the name of the compact model library to use.	<code>default.rsl</code>
<code>mapping=<Python_dict></code>	Sets the automatic translation from the name of the default SPICE model MOSFET to the RandomSpice <code><KEYWORD></code> .	<code>none</code>
<code>rsm_method=<string></code>	Selects the response surface model method. Options are: <ul style="list-style-type: none"><code>interp</code>: Use barycentric linear interpolation.<code>rsm</code>: Use the quadratic response surface model.	<code>interp</code>

Variability Section

The arguments available under the `Variability` section are listed here.

Argument	Description	Default
<code>process=<string></code>	Specifies whether to activate process variability in simulations. Not all libraries available with RandomSpice contain data on process variations. Therefore, the process is switched off by default. If it is activated for a compact model library that does not contain process variability information, only statistical variability information is applied. You can determine whether a compact model library contains process variation data by using the <code>-q</code> command-line option (see Starting RandomSpice on page 8).	False
<code>process_resp_only=<string></code>	Specifies whether to activate only the response surface component of process variability (switch off process variation generators if added to the library).	False
<code>statistical=<string></code>	Specifies whether to activate statistical variability in simulations.	True

Advanced Section

The arguments available under the `Advanced` section are listed here.

Argument	Description	Default
<code>spool=<string></code>	Specifies whether to activate spooling of internal simulation data.	True
<code>spoollimit=<string></code>	Sets the default size to limit data stored in memory.	1GB

It is usually not necessary to change the arguments in this section for most simulations performed with RandomSpice. However, they are provided for use in situations that require unusual simulation conditions.

To manage the amount of memory used by RandomSpice, which can become very large for large statistical ensembles, data stored internally is spooled to disk by default when it exceeds a certain size. Deactivating this can increase simulation speed, but it drastically increases the memory footprint for large simulations.

Chapter 2: Input Files

Structure of Input Files

The size limit can be set using `spoollimit`, which accepts values in bytes, KB, MB, and GB. This size limit applies to both generated netlists and output data. Therefore, RandomSpice uses approximately two times this value for storing internal data.

Summary of Input File Sections

Table 3 Summary of input file sections with command-line overrides if available

Input file section	Argument	Description	Default	Override
Circuit	netlist	File name of the template SPICE netlist	./in.net	-t <string>
Circuit	number	Number of randomized circuits to simulate	1	-n <integer>
Circuit	seed	Value used to seed the random number generator	0	-r <integer>
Circuit	startnum	Number of the first circuit in the statistical ensemble	1	-c <integer>
Simulation	enabled	Specifies whether SPICE simulations are run	True	-g
Simulation	mode	Simulation mode to use in RandomSpice	normal	-m <string>
Simulation	spice	SPICE simulator to use	hspice	-s <string>
Simulation	spiceargs	Additional arguments to be passed to SPICE simulator	-	-
Output	backend	Specifies how to store output data	-	-
Output	dir	Output directory in which to save results	./	-d <string>
Output	prefix	Prefix for output files	out	-o <string>
Output	processor	File name of Python postprocessing module to use	none	-
Output	savenets	Specifies whether to save generated netlists	False	-

Chapter 2: Input Files

Autogenerating Input Files

Table 3 Summary of input file sections with command-line overrides if available

Input file section	Argument	Description	Default	Override
Models	library	File name of compact model library to use	default.rsl	-L <string>
Variability	process	Specifies whether to activate process variability in simulations	False	-
Variability	process_resp_only	Specifies whether to activate only response surface component of process variability	False	-
Variability	statistical	Specifies whether to activate statistical variability in simulations	True	-
Advanced	spool	Specifies whether to spool internal simulation data	True	-
Advanced	spoollimit	Default size to limit data stored in memory	1GB	-

Autogenerating Input Files

When setting up a new simulation, you can use the `-i` (or `--gen-input`) command-line option to generate a template input file. The template is then written to the file name specified for RandomSpice, and any command-line overrides are incorporated into the template.

For example, to create an input file with all defaults set, specify the following on the command line:

```
$ RandomSpice2 template.rs2 -i
=====
TCAD to SPICE - RandomSpice

Version T-2022.03 for linux64

Copyright (c) 2010-2022 Synopsys, Inc.
This software and all associated documentation are proprietary to
Synopsys, Inc. and may only be used pursuant to the terms and conditions
of a written license agreement with Synopsys, Inc. All other use,
reproduction, modification, or distribution of this software
or the associated documentation is strictly prohibited.
=====
```

Chapter 2: Input Files

Autogenerating Input Files

Writing template input file to: template.rs2

```
$ cat template.rs2
[circuit]
seed = 0
startnum = 1
number = 1
netlist = in.net

[simulation]
spiceargs =
enabled = True
mode = normal
spice = ngspice

[output]
prefix = out
savenets = False
processor = none
dir = ./
backend = file

[models]
library = default.rsl
```

If you know some arguments that you want to set, these can be given at the same time as the `-i` option. For example:

```
$ RandomSpice2 sram.rs2 -i -t bitcell.net -r 1234 -n 100
-L GSS-25nm-ModelGen.rsl -d wm -o bc_wm
=====
TCAD to SPICE - RandomSpice

Version T-2022.03 for linux64

Copyright (c) 2010-2022 Synopsys, Inc.
This software and all associated documentation are proprietary to
Synopsys, Inc. and may only be used pursuant to the terms and conditions
of a written license agreement with Synopsys, Inc. All other use,
reproduction, modification, or distribution of this software
or the associated documentation is strictly prohibited.
=====

Writing template input file to: sram.rs2

$ catsram.rs2
[circuit]
seed=1234
startnum=1
number=100
netlist=bitcell.net

[simulation]
```

Chapter 2: Input Files

Preparing a Netlist

```
spiceargs=
enabled=True
mode=normal
spice=ngspice

[output]
prefix=bc_wm
savenets=False
processor=none
dir=wm
backend=file

[models]
library = GSS-25nm-ModelGen.rsl
```

The input file can then be further edited as required.

Preparing a Netlist

RandomSpice uses keywords to identify the transistors in a circuit that must be randomized. Keywords are used in place of the model identifier in a typical SPICE MOSFET statement:

```
M1 D G S B <keyword> L=25n W=25n
```

The keywords to be used vary depending on the compact model library. The keywords supported by a particular library are listed in the documentation for that library or can be found by using the RandomSpice `-q` command-line option (see [Starting RandomSpice on page 8](#)).

For example, common keywords supported by most libraries are:

- `ATOM:N` – atomistic NMOS device
- `ATOM:P` – atomistic PMOS device
- `UNIF:N` – uniform TCAD NMOS device
- `UNIF:P` – uniform TCAD PMOS device

To use a particular device or variability type, the keyword must be substituted. For example, the following netlist shows how a simple CMOS inverter netlist can be modified to operate with RandomSpice:

```
* CMOS Inverter

M1 Y A 0 0 ATOM:N L=25n W=25n
M2 Y A VDD VDD ATOM:P L=25n W=50n

VDD VDD 0 1
V1 A 0 PULSE(0 1 0 10p 10p 1n 2n)
```

Chapter 2: Input Files

Preparing a Netlist

```
.TRAN 10p 10n
.PRINT TRAN V(A) V(Y)

.OPTIONS NOPAGE=1 INGOLD=2

.END
```

Keyword replacement can be handled automatically through the `mapping` argument of the `Models` section in the RandomSpice input file, where you can supply a dictionary that maps default netlist keywords to RandomSpice library keywords (see [Models Section on page 23](#)).

Subcircuit Randomization

You can also apply randomization at the subcircuit level by appending `:RAND` to the subcircuit name. In this case, a new instance of the subcircuit is created with a new set of random transistors whenever the subcircuit is used.

For example, the following subcircuit uses a subcircuit definition to instantiate two inverters. Without gate-level randomization, this would result in both inverters using the same set of random transistors. As shown here, `:RAND` is appended to the subcircuit name, which is detected by RandomSpice and results in the output shown below the netlist:

```
* CMOS Buffer

.SUBCKT INV:RAND A Y VDD VSS
M1 Y A VSS VSS ATOM:N L=25n W=25n
M2 Y A VDD VDD ATOM:P L=25n W=50n

.ENDS

X1 A Y VDD 0 INV:RAND
X2 Y Z VDD 0 INV:RAND

VDD VDD 0 1
V1 A 0 PULSE(0 1 0 10p 10p 1n 2n)

.TRAN 10p 10n
.PRINT TRAN V(A) V(Y) V(Z)

.OPTIONS NOPAGE=1 INGOLD=2

.END

* CMOS Buffer
* Model Library: GSS 25nm HP Metal - LUT
* Random Seed: 1000

X1 A Y VSS 0 INV_1161036348612429450
```

Chapter 2: Input Files

Plug-in Postprocessing Modules

```
.SUBCKT INV_1161036348612429450 A Y VDD VSS
M1 Y A VSS VSS NMOS_W1_0_N468 L=2.5e-08 W=2.5e-08
M2 Y A VDD VDD PMOS_W2_0_N535 L=2.5e-08 W=5e-08
.ENDS
X2 Y Z VSS 0 INV_7913237767432230989
.SUBCKT INV_7913237767432230989 A Y VDD VSS
M1 Y A VSS VSS NMOS_W1_0_N670 L=2.5e-08 W=2.5e-08
M2 Y A VDD VDD PMOS_W2_0_N100 L=2.5e-08 W=5e-08
.ENDS

VDD VDD 0 1
V1 A 0 PULSE(0 1 0 10p 10p 1n 2n)

.TRAN V(A) V(Y) V(Z)
.PRINT TRAN V(A) V(Y) V(Z)

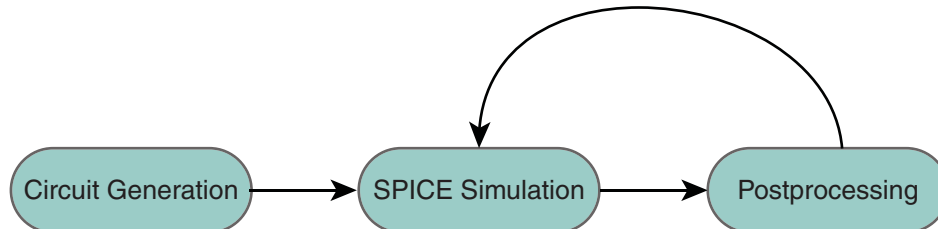
.OPTIONS NOPAGE=1 INGOLD=2

.model NMOS_W1_0_N670 NMOS
...
.END
```

Plug-in Postprocessing Modules

RandomSpice can postprocess the results of each SPICE simulation as part of its own simulation workflow. [Figure 2](#) shows the simulation workflow in the default normal mode. As detailed in [Simulation Modes on page 16](#), in normal mode, all circuits are generated first and then simulated, and postprocessing is performed in turn for each circuit. This mode allows you to define how the output data from the SPICE simulator is processed. For example, the postprocessing step can be used to perform text processing and calculations on the output.

Figure 2 Normal simulation mode of RandomSpice



To use this feature, you must provide your own Python module that performs the necessary processing and returns the relevant data to RandomSpice. User-supplied postprocessing modules must define a function called `Process`, which accepts two arguments, as follows:

```
def Process(data, **extra):
    # Do stuff
    return processed_data
```

The returned data is written to the output files created by RandomSpice and can be any numeric or textual data.

The arguments to the `Process` function are:

- `data` is a string containing the screen output from the SPICE simulator.
- `**extra` is a Python dictionary containing supplementary data sent from RandomSpice. This dictionary contains several items. The first is `cid`, which is the circuit number in the ensemble. The seed used for the circuit simulation is also passed in with the keyword `seed`. These can be accessed as follows:

```
def Process(data, **extra):
    circuit_number = extra['cid']
    seed = extra['seed']
    print("Circuit #: {0}".format(circuit_number))
    print("Seed for this circuit is: {0}".format(seed))
    # Further processing...
```

`**extra` also contains a `new_results` object, which includes auto-harvested PrimeSim HSPICE `.measurement`, `.print`, and waveform data.

For more information about the `new_results` object, from Sentaurus Workbench, choose **Help > Python API Documentation > Mystic**, and search for *HSPICESimulationResult*.

Python Library Modules

Some Python library modules are available for use in custom postprocessing modules, as well as some additional modules:

- `math`: Standard Python maths library
- `numpy`: Library for efficient handling of n -dimensional arrays and other numeric routines
- `re`: Python regular expressions library
- `scipy`: Library providing many routines for scientific data processing and numeric manipulation
- `string`: Python string processing library

3

Compact Modeling Strategies

This chapter describes the compact modeling strategies.

Subcircuit Device Models

RandomSpice supports generators for generic subcircuit models, effectively providing the following:

- Support for MOSFET subcircuit or macro models with external parasitic components. This allows the inclusion of middle-end-of-line variations using the RandomSpice generators, as well as layout-dependent effects, if these are characterized in a Mystic extraction. The following example shows a subcircuit model:

```
.subckt submosn d g s b w=1e-6 l=@lgate@

rd d d_i "40*1e-6/w"
rs s s_i "40*1e-6/w"

mos d_i g s_i b m28 l=1 w=w
.model m28 nmos
+vt0 = 0.4
+keta = -0.047
+ags = 0
+vsat = 100000
+etab = -0.004
+delta = 0.01
+pclm = 0.02
+pvag = 1
+rdswmin = 10
```

- Extending RandomSpice generator methodology beyond four-terminal MOSFETs to generic devices, which are supported by Mystic.

Supporting subcircuit models requires the Mystic extraction to be performed on a subcircuit model similar to the example shown, as well as through the SWB Python Library Builder, where the `Subcircuit` flag must be set up correctly. Finally, RandomSpice netlists must now be adjusted to instantiate models using “x” subcircuit notation instead of standard MOSFET “m” notation.

This is shown in the following example:

```
xmn vdrain_n gs vsource_n vbulk_n N49155:N
+ L=3.2e-08 DOEPAR1=1.0 DOEPAR2=0.8
+ w=1e-6
xmp vdrain_p gs vsource_p vbulk_p N51228:P
+ L=3.2e-08 DOEPAR1=1.0 DOEPAR2=0.8
+ w=1e-6

.data vddvals vddval 0 0.05 1.0
.DC vgs '-1.1*VDNOM' '1.1*VDNOM' 0.01 DATA=vddvals
.print i(vvdrain_n) i(vvdrain_p) lx82(xmn.mos) lx82(xmp.mos)
```

Response Surface Models

The device designs-of-experiments (DoEs) simulated as part of the TCAD to SPICE workflow can be handled in different ways, which depend on the final intended usage of the SPICE models:

- For general device improvement tasks over a very wide DoE, the recommended approach is to extract separator *point models* for each device in the DoE. This allows for extremely accurate compact modeling because devices are treated as effectively independent.
- The response surface model (RSM) approach is suited to cases where the DoE axes represent process variations and, in this case, a comprehensive SPICE model is extracted for the central point of the DoE, and then a subset of the SPICE model parameters is extracted to capture the device performance variation across the DoE.

RandomSpice libraries can include a simple RSM. This methodology allows for the representation of long-range process, or global variability, or capture device performance across a range of geometry, process, or any generic parameter related to the device that is characterized. This includes critical dimension (CD) parameters such as gate length (L_g), fin thickness (W_{fin}), and fin height (H_{fin}) for FinFETs, diameter (D) for circular nanowires, or silicon thickness (T_{Si}) for FDSOI devices.

Although standard compact models can model CD variation effects, they cannot model each of these effects without extensive iterative extraction. In some cases, the built-in compact model equations cannot capture the CD dependence across a range of technology options. Furthermore, the generic RSM approach allows the simple modeling of variation across process conditions that the compact model might be unable to capture directly, such as the halo implantation angle, doping profile variation, or variations related to etch or bake effects.

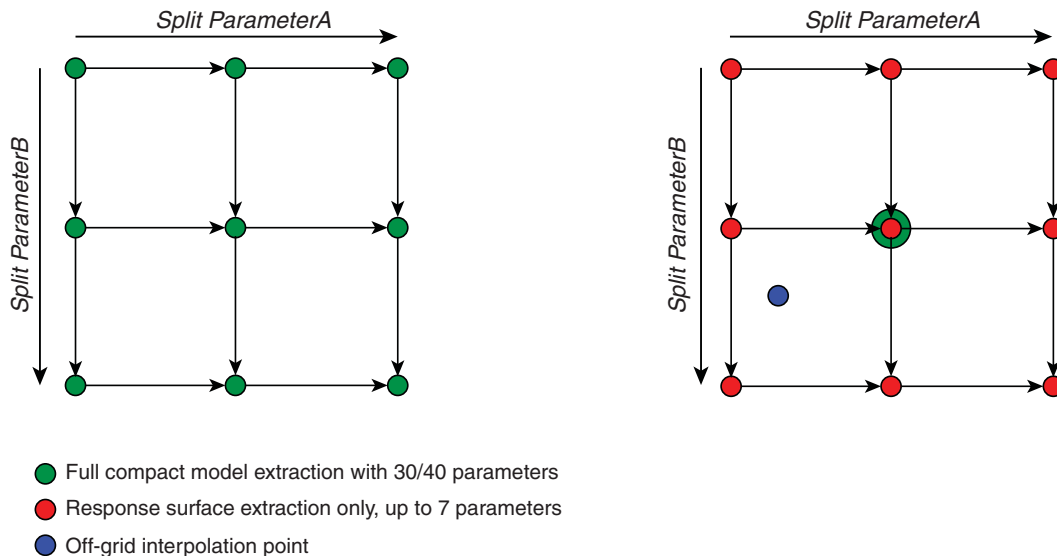
Figure 3 ((Left) Point model extraction approach where each point in the DoE is extracted as a separate device with very high accuracy and (right) RSM approach where full extraction is performed in the central point of the DoE and this model is used as the basis for extraction of a subset of response surface parameters (the RSM approach sacrifices some accuracy, but allows for off-grid interpolation))

Point model approach:

- Full compact model extraction at each DoE point
- Use initial model as starting point for each extraction
- Optimal accuracy for each point
- Cannot simulate off-grid (response surfaces not always reliable with this many parameters)

Response surface model approach:

- Full compact model extraction at central DoE point
- Response surface extraction using reduced parameter at each point
- Base model is the fully fit model at the central DoE point



Statistical Compact Models

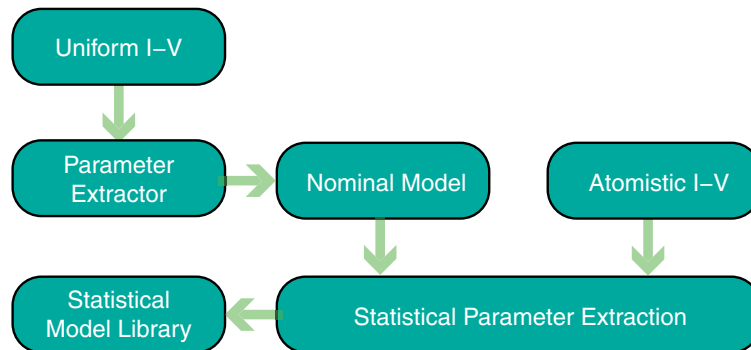
Accurate statistical compact models (SCMs) are the backbone of statistical circuit and system simulation and verification, providing vital information about the impact of variability on design and allowing performance–power–yield (PPY) optimization. Advanced SCM extraction techniques allow you to generate flexible and accurate SCMs.

SCM parameter sets that guarantee accurate results are selected carefully, based on comprehensive sensitivity analyses and the physical consideration of the impact of individual variability sources.

When extracting SCMs for digital applications, gate current–voltage characteristics at both high-drain and low-drain bias provide sufficient variability information for the accurate description of statistical circuit-switching behavior. To provide the best SCMs available, these characteristics are used as fitting targets during direct statistical parameter extraction,

and multistage local parameter statistical extraction strategies are used that preserve the physical meaning of the fitting parameters.

Figure 4 Statistical extraction methodology



This advanced direct extraction approach does not require that the variation in device electrical performance follow any particular distribution, and it does not make any *ab initio* assumptions about the distribution and correlation of SCM parameters. As a result, this approach provides the most accurate representation possible of device characteristics obtained from 3D Garand VE simulations or from measurement, and preserves critical high-moment information and parameter correlation.

Directly extracted SCM parameters can also form the basis for accurate SCM generation on the fly by using principal component analysis (PCA) and ModelGen.

Gaussian Parameter Generation

A common approach to SCM parameter generation is to treat parameters as uncorrelated Gaussian-behaved distributions. Average and standard deviation values can be extracted from the SCM ensembles to provide a first-order approximation to the true distributions.

Since the distributions of compact models are frequently nonnormal distributed, SCM approaches based on the assumption of uncorrelated normal distributions of the SCM parameters could introduce considerable errors in statistical circuit simulation.

More advanced approaches to parameter generation include principal component analysis (PCA), which can account for parameter correlations, and the nonlinear power method, which can account for both parameter correlations and nonnormal distributions.

In addition to Gaussian parameter generation, both of these methods are implemented in RandomSpice (see [Principal Component Analysis on page 36](#) and [ModelGen Approach on page 37](#)).

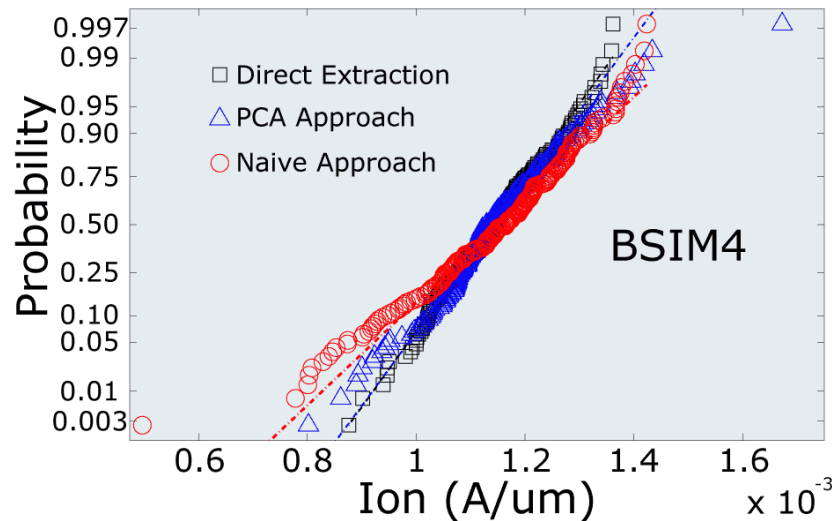
Principal Component Analysis

The SCM parameters directly extracted from target sets of current–voltage characteristics are correlated and might have distinctly nonnormal distributions. The correlations between extracted statistical parameters can be treated correctly at the statistical parameter generation stage using principal component analysis (PCA).

PCA transforms the set of correlated SCM parameters into a set of uncorrelated random variables (*principal components*). These principal components can then be generated independently and transformed back into SCM parameter sets preserving the original correlations between the original parameters [1].

The SCM extraction suite can generate channel length–dependent and channel width–dependent principal components, and the corresponding transformation matrices that allow the generation of properly correlated statistical parameter sets for transistors with arbitrary geometries. Then, RandomSpice can use the corresponding SCM libraries to perform statistical circuit simulations, preserving the correlation between the extracted compact model parameters and guaranteeing the statistical accuracy of the simulation results.

Figure 5 Comparison of direct extraction, Gaussian (Naive), and PCA approaches to SCM parameter generation

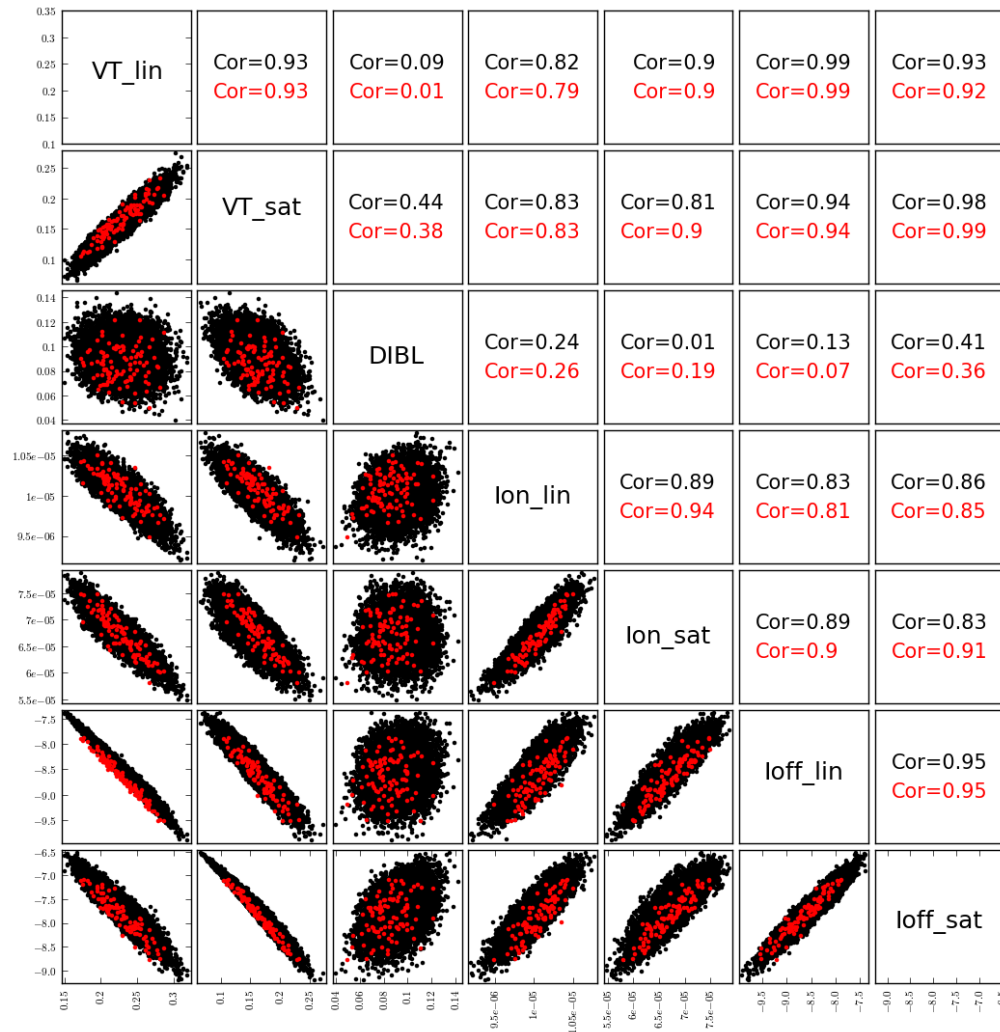


PCA assumes that the originally correlated SCM parameters are normally distributed and works well if their distributions are close to normal in reality. If the extracted compact model parameter distributions strongly deviate from normal distributions, even the use of PCA can result in significant errors in the statistical circuit simulation, and it is necessary to use the nonlinear power method described below.

ModelGen Approach

Directly extracted SCM parameters do not always follow normal distributions as typically assumed by standard PCA. This can introduce significant errors in statistical circuit simulations, particularly when the impact of rare devices in the tails of distributions is critical in determining yield. Advanced statistical tools capable of capturing and analyzing higher order moments in statistical parameter distributions become vital to the analysis of circuits with large populations of identical transistors such as SRAMs, DRAMs, and flash memories.

Figure 6 Correlation plot between transistor figures of merit: red points show 200 statistical devices simulated in TCAD and extracted using SCM approach, and black points show 10000 SPICE devices generated using ModelGen approach



Chapter 3: Compact Modeling Strategies

References

To fulfill this need, RandomSpice offers the ModelGen approach for SCM parameter generation, which accurately reproduces the shapes and tails of nonnormally distributed SCM parameters, while ensuring that correlations between these parameters are maintained.

The advanced ModelGen approach is a significant step toward the goal of developing a completely general SCM parameter generation methodology.

References

- [1] B. Cheng *et al.*, “Benchmarking the Accuracy of PCA Generated Statistical Compact Model Parameters Against Physical Device Simulation and Directly Extracted Statistical Parameters,” in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, San Diego, CA, USA, pp. 143–146, September 2009.

4

Generating RandomSpice Libraries

This chapter describes how to generate RandomSpice libraries.

Library Builder

You can use the RandomSpice Library Builder to generate RandomSpice libraries. The default usage case assumes that all extraction data is stored in the TCAD to SPICE database. The basic functionality of the Library Builder includes the following:

- Adds base model cards to the library, tied to a device label
- Builds response surface model (RSM) parameter sets on top of the base model for a single device
- Adds local-variability parameter distributions, to either a base model or an RSM
- Adds global variability generators based on simple Gaussian distributions and correlation coefficients

The Library Builder is documented in the TCAD to SPICE API Reference (from Sentaurus Workbench, choose **Help > Python API Documentation > TCAD to SPICE**).

Library Builder Example

This example demonstrates an RSM design-of-experiments (DoE) with local variability.

Prerequisites

- Mystic uniform extraction for the base model card
- Mystic RSM extraction
- Mystic local variability extraction for all splits in a Sentaurus Workbench project underpinned by RandomSpice

Chapter 4: Generating RandomSpice Libraries

Library Builder Example

The first part of creating any RandomSpice library is to add the base model card, which represents a comprehensive Mystic extraction at the nominal DoE conditions. In the following example, you create the device names `nfet` and `pfet` based on the Sentaurus Workbench device type (`nMOS` and `pMOS`).

First, you find the database projects with the Sentaurus Workbench tool label `Mystic_Uniform`, and then you add the final fitted model to the `builder` object by using the `add_device` function:

```
remap_names={"nMOS":"nfet","pMOS":"pfet"}

#####
### The following code is for the midpoint of the DoE only. ###
### These base model cards will be the basis for the          ###
### response surface model and the variability model.          ###
#####

midpoints = {p.metadata["swb"][:p for p in dbi.get_project(
    midpoint=True,
    swb__tool_label="Mystic_Uniform")}]
for proj in midpoints.values():
    remap_name=remap_names[proj.metadata["swb"][:p]]
    # Add base model card
    builder.add_device(remap_name, dev_type=proj.metadata["swb"][:p],
                      dataset=list(proj.datasets)[-1], point_model=False)
```

At this point, the RandomSpice library contains a base model card (for both NMOS and PMOS) under the device labels `nfet:N` and `pfet:P`. RSM points can be added easily to the library using the following code, where `rs_projects` are all the Sentaurus Workbench nodes where the tool label is `Mystic_Response_Surface`. The RSM data is added to the device based on the remap name for that device type.

```
#####
### The following code adds response surface models          ###
### corresponding to each DoE point.                          ###
#####
rs_projects = dbi.get_project(
    swb__tool_label="Mystic_Response_Surface")
for proj in rs_projects:
    dev_midpoint = remap_names[proj.metadata["swb"][:p]]
    # Add RSM model points
    builder.add_doe_point(dev_midpoint, proj.metadata["doe"],
                        dataset=proj.datasets[-1],
                        midpoint=proj.metadata["midpoint"])
```


Chapter 4: Generating RandomSpice Libraries

Library Builder Example

Finally, local variability data is added in a similar way to the RSM data, by iterating through all the Sentaurus Workbench nodes where `Mystic_Statistical_Extraction` is the tool label.

```
#####
### The following code adds local variability models ###
### corresponding to each DoE point. ###
#####
stat_projects = dbi.get_project(tool_label="Mystic_Statistical")

for proj in stat_projects:
    dev_midpoint = remap_names[proj.metadata["swb"]]["Type"]
    # Add local-variability model points
    builder.add_lv_distributions(dev_midpoint, proj.metadata["doe"],
                                dataset=proj.datasets[-1],
                                midpoint=proj.metadata["midpoint"])
```

Finally, the library is built using the `build` command. The `debug` flag activates the verbose builder mode, which prints to screen all RSM and local variability parameters that characterize the resultant RandomSpice library, as well as the DoE axes and expectant model instance parameters required in the RandomSpice-compatible SPICE netlist.

```
# Save the library
builder.build("@pwd@/@nodedir@", libname="{0}".format("n@node@_library"),
              doe=True, gv_dist=True, lv_dist=True, debug=True,
              non_unif_gv=False)
```

Glossary

CD

Critical dimension.

DoE

Design-of-experiments.

DTCO

Design-technology cooptimization.

PCA

Principal component analysis.

PPY

Performance–power–yield.

RMS

Root mean square.

RSM

Response surface model.

SCM

Statistical compact model.

SNM

Static noise margin.

Index

A

Advanced section 20, 24
autogenerating input files 26

B

barycentric linear interpolation 23
batch simulation mode 16
build command 41

C

Circuit section 20
CMOS inverter 28
command-line options 8
compact model library 23, 24
critical dimension variation effects 33

D

data storage 22
debugging 11

F

first-order approximation 35

G

Gaussian parameters
 generating 35
generate circuits 8

H

hybrid input files 19

I

input files 19
 Advanced section 20, 24
 autogenerating 26
 Circuit section 20
 hybrid 19

Models section 20, 23
Output section 20, 22
 overriding arguments 9
Simulation section 16, 20, 21
Variability section 20, 24
interconnect variability 16

L

Library Builder 16, 32, 39

M

Mersenne Twister random number generator 21
ModelGen 16, 37
Models section 20, 23
multiprocessors 18
Mystic extraction 32

N

netlists 11
 CMOS inverter 28
 preparing 28
new_results object 31
normal simulation mode 16, 30

O

Output section 20, 22
overriding arguments in input files 9

P

parallel processing 18
performance–power–yield (PPY) optimization 34
point models 33
 extracting 34
postprocessing 30
PrimeSim HSPICE tool 7
principal component analysis 16, 35, 36
process variability 24
Python library modules 31

Index

Python postprocessing module 23

R

random number generator 9, 21

RandomSpice

- libraries 33

- libraries, generating 39

- starting 8

S

sequential simulation mode 16

simulation modes 16, 21

simulation procedure 11

Simulation section 16, 20, 21

SPICE simulators 22

spooling 16, 24

statistical compact models 34

statistical variability 24

storing data 22

subcircuit models 32

subcircuits

- randomization 29

T

TCAD to SPICE workflow 33

template netlist 9

transistors

- identifying 28

V

Variability section 20, 24