

# **Virtuoso Analog Design Environment GXL User Guide**

**Product Version 6.1.5**

**June 2013**

© 1999–2013 Cadence Design Systems, Inc. All rights reserved.

Printed in the United States of America.

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

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

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

All other trademarks are the property of their respective holders.

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

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

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

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

# Contents

---

<b>Preface</b> .....	13
<u>ADE GXL Product Features</u> .....	13
<u>Licensing in Analog Design Environment (ADE) GXL</u> .....	13
<u>Related Documents for ADE GXL</u> .....	13
<u>Third Party Tools for ADE GXL</u> .....	14
<u>Typographic and Syntax Conventions</u> .....	14
 <b>1</b>	
<b>Getting Started in the Environment</b> .....	17
<u>Launching the ADE GXL Environment</u> .....	18
<u>Using the ADE GXL Banner Menu</u> .....	18
 <b>2</b>	
<b>Sensitivity Analysis</b> .....	19
<u>Generating Sensitivity Data</u> .....	20
<u>Support for Multi-Technology Simulations in Sensitivity Analysis</u> .....	26
<u>Troubleshooting a Design Point from the Sensitivity Results</u> .....	27
<u>Specifying Variables, Parameters, and Model Files to be Varied for Sensitivity Analysis</u>	
28	
<u>Selecting Instances and Parameters for Mismatch Variation</u> .....	33
<u>Saving Sensitivity Data for DC Operating Point Parameters</u> .....	40
<u>Viewing Sensitivity Data</u> .....	54
<u>Reading Data in the Sensitivity Analysis window</u> .....	55
<u>Viewing Sensitivity Data in Different Tabs</u> .....	56
<u>Working with Sensitivity Data</u> .....	60
<u>Changing the Type and Format of Data To View</u> .....	60
<u>Changing the Data View Type</u> .....	71
<u>Filtering Data Based on Correlation Coefficient</u> .....	72
<u>Filtering Data Based on Columns</u> .....	72
<u>Filtering Data Based on Search Criteria</u> .....	74
<u>Resetting Filters</u> .....	75

<u>Sorting Data</u>	75
<u>Hiding Corner Data</u>	76
<u>Hiding Specification Data</u>	76
<u>Hiding or Showing the Detailed Results for Corners</u>	76
<u>Hiding or Showing the Detailed Results for Specifications</u>	77
<u>Hiding or Showing Statistical Parameters</u>	77
<u>Showing All Hidden Data</u>	77
<u>Highlighting Associated Devices in the Schematic</u>	77
<u>Viewing Results in Multiple Sensitivity Analysis Windows</u>	78
<u>Changing Number Format</u>	78
<u>Plotting Sensitivity Analysis Results</u>	80
<u>Setting the Plotting Mode</u>	84
<u>Displaying the ADE XL and Sensitivity Analysis Results in the Same Graph</u>	90
<u>Plotting Average Sensitivity Analysis Results</u>	92
<u>Displaying Spec Markers in Sensitivity Analysis Plots</u>	93
<u>Saving the Sensitivity Data</u>	95
<u>Creating Worst Case Corners</u>	96
<u>Specifying the Setup</u>	96
<u>Creating Worst Case Corners</u>	103
<u>Viewing Worst Case Corners</u>	103
<u>Creating Worst Case Corners Interactively</u>	105
<u>Viewing Mismatch Analysis Results</u>	108
<u>Changing the Type of Results to be Displayed</u>	112
<u>Changing the Number of Device Parameters</u>	112
<u>Filtering the Data</u>	113
<u>Viewing Associated Devices</u>	113
<u>Saving the Mismatch Data</u>	113
<u>3 Circuit Optimization</u>	115
<u>Parameterizing the Design</u>	116
<u>Matching Devices and Device Properties</u>	116
<u>Defining Values for Optimization</u>	120
<u>Setting Up Specifications</u>	122
<u>Specifying General Specifications</u>	122

<u>Specifying Area Constraint Specifications</u>	125
<u>Running Optimization</u>	130
<u>Running a Local Optimization</u>	131
<u>Running a Global Optimization</u>	134
<u>Viewing the Variable Data from Optimization Results</u>	136
<u>Manual Tuning</u>	137
<u>Sizing Over Corners</u>	143
<u>Running Feasibility Analysis</u>	148
<u>Improving the Yield</u>	151
<u>Creating, Viewing, and Modifying Reference Points</u>	157
<u>Creating a Reference Point from Scratch</u>	157
<u>Creating, Viewing, and Modifying a Reference Point from a Run</u>	159
<u>Viewing and Modifying the Current Reference Point</u>	161
<u>Specifying How Much Optimization Data to Save</u>	163
<u>Data Points—Definition</u>	165
<u>Design Points—Definition</u>	165

## 4

<u>High Yield Estimation</u>	167
<u>Performing High Yield Estimation</u>	168
<u>High Yield Estimation with Multiple Corners</u>	176
<u>Creating Statistical Corners From a Worst Case Distance Analysis</u>	179

## 5

<u>Multi-Technology Simulation</u>	185
<u>Enabling Multi-Technology Simulation</u>	186
<u>Specifying MTS Options</u>	187
<u>Viewing Cells by Instance Name</u>	188
<u>Limiting the View by Library or Instance</u>	190
<u>Specifying a Cell as an MTS Block</u>	190
<u>Specifying Simulator Options for an MTS Block</u>	191
<u>Specifying a Model Library for an MTS Block</u>	191
<u>Specifying Test and Block Information when Adding Model Files to Corners</u>	192
<u>Disabling MTS for Third-Party Simulators</u>	192
<u>Detecting Context Symbol/Name Collisions</u>	194

## 6

<u>Design Characterization and Modeling</u> .....	197
<u>DCM Process Flows</u> .....	199
<u>Bottom-Up Modeling</u> .....	199
<u>Top-Down Modeling</u> .....	199
<u>Liberty MS Model Generation</u> .....	200
<u>Verilog-A[MS] Model Calibration</u> .....	200
<u>DCM Developer</u> .....	200
<u>Design Verification</u> .....	201
<u>Setting Up VHDL-AMS and Other Model Formats</u> .....	202
<u>Setting Up the Create Block in the .gui File</u> .....	202
<u>Specifying Environment Variables</u> .....	203
<u>Launching DCM</u> .....	204
<u>Accessing the Start Form</u> .....	205
<u>Opening a Local DCM Data File</u> .....	207
<u>Opening a Recently-Opened DCM Data File</u> .....	207
<u>Browsing for a DCM Data File to Load</u> .....	207
<u>Creating a New DCM Data File</u> .....	208
<u>Saving Changes</u> .....	209
<u>Using the Tools Menu</u> .....	210
<u>Viewing the Last Error</u> .....	211
<u>Viewing Generated Files</u> .....	212
<u>Viewing the Generation Log File</u> .....	214
<u>Viewing Characterization and Modeling Details</u> .....	215
<u>Viewing Any Text File</u> .....	217
<u>Viewing Bottom-Up Verilog-A[MS] Generated Files</u> .....	218
<u>Viewing Bottom-Up VHDL-AMS Generated Files</u> .....	220
<u>Viewing Bottom-Up Verilog-D Generated Files</u> .....	222
<u>Viewing Bottom-Up Liberty Generated Files</u> .....	223
<u>Viewing Black Box Liberty (.lib) Files</u> .....	224
<u>Viewing Verification Files</u> .....	225
<u>Viewing Top-Down Verilog-A[MS] Generated Files</u> .....	227
<u>Viewing Top-Down VHDL-AMS Generated Files</u> .....	228
<u>Viewing the Results Database</u> .....	229
<u>Exiting Design Characterization and Modeling</u> .....	231

# 7

<b>Model Generation</b>	233
<b>Specifying a Design</b>	234
Selecting an Instance on the Schematic	234
Selecting a Library, Cell, and View	235
Typing the Library, Cell, View, and Pin Information	235
Opening a Design from the Design Tab	236
Selecting the Testbench Library	236
Opening the Testbench Schematic	236
Loading Model Setup and Simulator Options from an ADE State Directory	237
Loading ADE State	237
Loading from a Directory	240
<b>Specifying a Function</b>	241
Specifying Design Pin Types for Top-Down Modeling	242
Selecting a Simulator	243
Filtering Design Functions	243
Viewing Design Function Data	247
Enabling Differential Input/Output Pin Types	250
Specifying Additional Components between a Pin and Ground	250
Specifying Design Pin Attributes	251
Viewing Function Help	252
<b>Specifying Modeling Requirements</b>	253
Specifying Top-Down Modeling for AMS Models	253
Specifying Bottom-Up Modeling for AMS Models	259
Specifying Bottom-Up Modeling for Verilog-D	263
Specifying Bottom-Up Modeling for Liberty	265
<b>Setting Up Model Calibration Requirements</b>	268
Calibrating Models	269
Modifying Calibration Sweep Parameters	271
Specifying Parameters	278
Specifying Options	281
<b>Generating Models and Running Tests</b>	282
Generating Tests and Models	282
Running Tests	283
<b>Generating Liberty MS Models</b>	287

<u>Model Component Parameters</u>	290
<b>8</b>	
<b>Calibrating Verilog-A[MS] Models</b>	301
<u>Launching Model Calibration</u>	302
<u>Specifying the Source Model</u>	304
<u>Setting Up the Model</u>	305
<u>Specifying the Destination Model</u>	312
<u>Saving the Model Calibration Setup</u>	314
<u>Calibrating the Model</u>	315
<b>9</b>	
<b>Design Verification</b>	317
<u>Enabling Design Verification</u>	317
<u>Defining Specifications for Design Verification</u>	318
<u>Disabling a Test and Measurement</u>	319
<u>Updating Tests and Measurements from Function Information</u>	319
<u>Specifying Conditions for Design Verification</u>	320
<u>Adding Conditions</u>	321
<u>Enabling Conditions</u>	321
<u>Disabling Conditions</u>	322
<u>Renaming Conditions</u>	322
<u>Removing Conditions</u>	322
<b>10</b>	
<b>Cell Type Development</b>	323
<u>Understanding the dcmDeveloper Wizard</u>	325
<u>Launching dcmDeveloper to Create New Cell Types</u>	328
<u>Creating a New Cell Type in the dcmDeveloper Wizard</u>	331
<u>Setting Objectives</u>	331
<u>Specifying Design Information</u>	333
<u>Selecting the Models</u>	335
<u>Specifying Tests</u>	337
<u>Enable Pin Types</u>	343

# Virtuoso Analog Design Environment GXL User Guide

---

<u>Define Pin Type Rules</u>	345
<u>Calibrating the Verilog-A[MS] Model</u>	348
<u>Reviewing the Cell Type</u>	350
<u>Completing Tasks</u>	352
<u>Creating a New Cell Type</u>	356
<u>Adding and Configuring Test Benches</u>	357
<u>Loading an ADE XL View</u>	357
<u>Adding the Test Benches</u>	360
<u>Copying a Test</u>	362
<u>Renaming a Test</u>	363
<u>Deleting Tests</u>	363
<u>Adding a Measure</u>	365
<u>Adding Instance Text</u>	372
<u>Adding Pin Types to be Iterated</u>	373
<u>Adding a Condition</u>	374
<u>Moving Instances or Models into a Condition</u>	376
<u>Modifying an Instance</u>	377
<u>Adding and Modifying Pin Types</u>	379
<u>Automatically Replacing Pin Types</u>	386
<u>Adding and Modifying Parameters</u>	388
<u>Viewing the Test Instance</u>	395
<u>Loading State Files</u>	396
<u>Modifying the States</u>	396
<u>Adding Models</u>	398
<u>Create the Model</u>	398
<u>Adding a Model Calibration Statement</u>	400
<u>Templatizing Text</u>	403
<u>Specifying Model Options</u>	404
<u>Specifying Cell Type Generation Options</u>	414
<u>Setting Cell Type Information</u>	416
<u>Specifying Cell Creation Options</u>	416
<u>Specifying Defaults</u>	416
<u>Specifying the Output Destination</u>	418
<u>Adding and Modifying Pin Rules</u>	421
<u>Saving the dcmDeveloper Data</u>	425
<u>Closing dcmDeveloper</u>	425

<u>Viewing an Existing Cell Type</u>	426
<u>Opening from DCM</u>	426
<u>Opening from dcmDeveloper</u>	426
<b>A</b>	
<u>OpenDCM</u>	429
<u>OpenDCM File Types</u>	430
<u>OpenDCM Directory Structure</u>	431
<u>OpenDCM .gui File</u>	433
<u>function Block</u>	434
<u>odcm Block</u>	441
<u>create Block</u>	442
<u>parameters Block</u>	443
<u>Top-Down Verilog-A Block</u>	446
<u>Verilog-A Block</u>	448
<u>Verilog-D Block</u>	453
<u>Defaults Block</u>	454
<u>Liberty Block</u>	455
<u>Sweeps Block</u>	455
<u>OpenDCM Test Template</u>	455
<u>DCM_GENERATE</u>	457
<u>DCM_FOREACH Block</u>	459
<u>DCM_IF Block</u>	460
<u>DCM_DEFINE Statement</u>	462
<u>DCM_SKIP TEST Directive</u>	464
<u>COMPONENT Block</u>	464
<u>Continuation Character for Test and Model Template Files</u>	468
<u>OpenDCM Model Template</u>	468
<u>DCM_ANALOG</u>	470
<u>DCM_CALIBRATE</u>	471
<u>DCM_COMMENT</u>	474
<u>DCM_DEBUG</u>	474
<u>DCM_MODULE</u>	475
<u>DCM_NO_MODULE</u>	475
<u>DCM_SECTION</u>	475

<u>DCM VA MODULES</u>	476
<u>DCM_VARS</u>	476
<u>DCM_VERIFY</u>	476
<u>OpenDCM .txt Help File</u>	476
<u>OpenDCM Pin Type Substitutions</u>	477
<u>Pin Index</u>	478
<u>Pin versus String Notation</u>	479
<u>OpenDCM Functions</u>	479
<u>Design Location Access Functions</u>	480
<u>Pin Access Functions</u>	480
<u>Parameter Value Access Function</u>	485
<u>Sweep Value Access Functions</u>	485
<u>Verilog-A Value Access Functions</u>	485
<u>Verilog-D Value Access Functions</u>	488
<u>Liberty (.lib) Value Access Functions</u>	489
<u>Liberty (.lib) Model Calibration Functions</u>	489
<u>Simulator Access Functions</u>	497
<u>Special Functions and Statements</u>	497
<u>OpenDCM Perl Extensions</u>	502
<u>OpenDCM Perl Block</u>	502
<u>Iterating through Pin Names using Perl</u>	503
<u>Data Access Functions for Perl blocks</u>	504
<u>Debugging Blocks of Perl Code</u>	507
<u>OpenDCM Variables</u>	508
<u>OpenDCM Command and Macro Reference</u>	508
<u>Creating Templates That Support Differential Pins</u>	511
<u>The .gui File</u>	512
<u>The .ade File</u>	512
<u>The .va File</u>	513
<b>B</b>	
<u>Environment Variables</u>	515
<u>ADE GXL Environment Variables</u>	516
<u>sensitivityPlotContinuousLine</u>	516
<u>sensitivityThumbnailMaxPointsForLinePlot</u>	516

<u>digitsToShowForYieldInPercentage</u>	.....	517
<u>sortVariablesOpt</u>	.....	517
<u>stopManualTuningOnSessionExit</u>	.....	518
<u>Design Characterization and Modeling</u>	.....	519
 <b>C</b>		
<u>Form Descriptions</u>	.....	523
<u>Advanced Verilog-A</u>	.....	524
<u>Choose Design</u>	.....	525
<u>Choose a Directory</u>	.....	526
<u>Design Pin Attributes</u>	.....	527
<u>Function Filter</u>	.....	528
<u>Liberty (.lib) Library Generation</u>	.....	529
<u>Model Destination</u>	.....	530
<u>Run Status</u>	.....	532
<u>Start</u>	.....	533
<u>Sweep Setup</u>	.....	534
<u>Index</u>	.....	535

# Preface

---

You can perform optimization, characterization and modeling tasks, and multi-technology simulation on your designs in the ADE GXL environment. ADE GXL eliminates redundant data entry and simplifies moving between design tasks. You can also customize your user interface (see the [Virtuoso® Design Environment User Guide](#)).

## ADE GXL Product Features

From the ADE GXL environment, you can access all features of [ADE XL](#) and the following:

- [Set up and run optimizations](#)
- [Characterize design behavior and generate behavioral models](#)
- [Multi-technology simulation](#)
- [Set up and run parasitic resimulation](#)

## Licensing in Analog Design Environment (ADE) GXL

For information on licensing in the Virtuoso design environment, see [Virtuoso Software Licensing and Configuration Guide](#).

## Related Documents for ADE GXL

The following documents provide more information about the topics discussed in this guide.

- [Virtuoso Schematic Editor User Guide](#) describes Cadence's schematic editor.
- [Virtuoso Analog Design Environment XL User Guide](#) describes the ADE XL environment.
- [Virtuoso Parasitic Estimation and Analysis User Guide](#) describes the Cadence parasitic simulation product.
- [Spectre Circuit Simulator User Guide](#) and [Spectre Circuit Simulator Reference](#) describe Cadence's Spectre analog circuit simulator.

- *SpectreRF Simulation Option User Guide* describes Cadence's RF circuit simulation option.
- *UltraSim User Guide* describes Cadence's multi-purpose single engine, hierarchical simulator, designed for the verification of analog, mixed signal, memory, and digital circuits.
- *Virtuoso AMS Designer Simulator User Guide* describes Cadence's AMS mixed-signal circuit simulator.
- *Virtuoso Visualization and Analysis Tool User Guide* contains product information for waveform viewing and post-processing of simulation results.
- *Component Description Format User Guide* describes Cadence's Component Description Format (CDF) for describing parameters and the attributes of parameters of individual components and libraries of components.
- *Analog Expression Language Reference* contains concept and reference information about the Analog Expression Language (AEL).

## Third Party Tools for ADE GXL

To view any .swf multimedia files, you need:

- Access to the [Cadence Online Support](#) website.
- Flash-enabled web browser, for example, Internet Explorer 5.0 or later, Netscape 6.0 or later, or Mozilla Firefox 1.6 or later. Alternatively, you can download Flash Player (version 6.0 or later) directly from the [Adobe](#) website.
- Speakers and a sound card installed on your computer for videos with audio.

## Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates text you must type exactly as it is presented.
<i>z_argument</i>	Indicates text that you must replace with an appropriate argument. The prefix (in this case, <i>z_</i> ) indicates the data type the argument can accept. Do not type the data type or underscore.

## Virtuoso Analog Design Environment GXL User Guide

### Preface

---

[ ]	Denotes an optional argument. When used with vertical bars, they enclose a list of choices from which you can choose one.
{ }	Used with vertical bars, they denote a list of choices from which you must choose one.
	Separates a choice of options.
...	Indicates that you can repeat the previous argument.
=>	Precedes the values returned by a Cadence® SKILL language function.
/	Separates the possible values that can be returned by a Cadence SKILL language function.
<i>text</i>	Indicates names of manuals, menu commands, form buttons, and form fields.

# **Virtuoso Analog Design Environment GXL User Guide**

## Preface

---

---

# Getting Started in the Environment

---

When you launch the environment, *ADE GXL* and *Parasitics* appear on the menu banner.

The ADE GXL environment consists of a set of toolbars and assistant panes that make up your *workspace*. You can load a Cadence workspace or create and load a custom workspace. You can specify what workspace to load for a given cellview. For more information about workspaces, see “[Getting Started with Workspaces](#)” in the *Virtuoso Design Environment User Guide*.

The *Parasitics* menu lets you access Virtuoso® Parasitic Estimation and Analysis to investigate the effects of parasitics on your circuits. You can report on parasitics that exist in your design, show or hide them on your design, and create refined extracted cell views. For more information, see the *Virtuoso Parasitic Aware Design User Guide*.

See the following topics for more information about the ADE GXL environment:

- [Launching the ADE GXL Environment](#) on page 18
- [Using the ADE GXL Banner Menu](#) on page 18
- “[Specifying the Run Mode](#)” in the *Virtuoso® ADE XL User Guide*

See also

- [Chapter 3, “Circuit Optimization”](#) for information about circuit optimization
- [Chapter 5, “Multi-Technology Simulation”](#) for information about using multi-technology simulation
- [Chapter 6, “Design Characterization and Modeling”](#) for information about design characterization and behavioral model generation
- The *Virtuoso® Parasitic Estimation and Analysis User Guide* for information about parasitic resimulation

## Launching the ADE GXL Environment

To start the environment, do one of the following:

To open the environment from a schematic editing window

- In the schematic editing window, choose *Launch – ADE GXL*.

The environment appears.

*ADE GXL* and *Parasitics* appear on the menu banner.

**Note:** If you have descended into a design hierarchy, the environment returns you to the top level of your design when you choose *Launch – ADE GXL*.

To open an existing ADE GXL cellview from the CIW

1. In the CIW, choose *File – Open*.

The Open File form appears.

2. Select an ADE GXL design cellview.

The environment appears.

*ADE GXL* and *Parasitics* appear on the menu banner.

## Using the ADE GXL Banner Menu

When you launch the ADE GXL environment, *ADE GXL* appears on the menu banner. The menu entries available are the same as those for ADE XL.

**Note:** For information about the *Parasitics* menu, which also appears when you launch ADE GXL, see the *Virtuoso Parasitic Aware Design User Guide*.

---

## Sensitivity Analysis

---

Sensitivity analysis helps in finding out the sensitivity of specifications to variables, device parameters, statistical parameters and DC operating point parameters. This chapter provides an overview of sensitivity analysis and describes how to run this analysis and view results. It also explains how to create worst case corners based on the results of sensitivity analysis.

For more details, refer to the following sections:

- [Generating Sensitivity Data](#) on page 20
- [Viewing Sensitivity Data](#) on page 54
- [Working with Sensitivity Data](#) on page 60
- [Plotting Sensitivity Analysis Results](#) on page 80
- [Saving the Sensitivity Data](#) on page 95
- [Creating Worst Case Corners](#) on page 96
- [Viewing Mismatch Analysis Results](#) on page 108

## Generating Sensitivity Data

You can generate and view data on the sensitivity of specifications to variables, device parameters, statistical parameters and DC operating point parameters.

### ***Variation of Design Variables and Device Parameters***

ADE GXL varies each variable or parameter one-factor-at-a-time (OFAT). To calculate sensitivity, the tool perturbs the variable or parameter value around its nominal value, while leaving other variables and parameters as fixed.

For design and PVT variation, ADE GXL provides two methods:

- OFAT 3-level: In this method, you can specify three values for a variable. For example, 1.9u, 2.0u, 2.1u with nominal value equal to 2.0u. If, instead of three points, you specify a range of values, the tool chooses the following values of the variable for simulation:
  - one step below the nominal value
  - nominal value
  - one step above the nominal value
- For example, if the variable range is 1u:0.1u:5u and the nominal value is 3.6u, the three values used by the tool are: 3.5u, 3.6u, and 3.7u.
- OFAT Sweep: In this method, you can specify more than three values. For example, if you specify the variable range as 200n:50n:400n, and the nominal value as 250n, the variable values used for simulation are: 200n, 250n, 300n, 350n, and 400n.

### ***Variation of Statistical Parameters***

For statistical variation, ADE GXL varies each statistical parameter one-factor-at-a-time at 3 levels (values). When the statistical parameter is modeled with normal or log normal distribution, ADE GXL uses the following values for simulation:

- N sigma below the mean value
- mean value
- N sigma above the mean value

When the statistical parameter is modeled with uniform distribution, ADE GXL chooses values based on the percentage of range.

Sometimes the variable whose sensitivity is being determined is at its boundary of possible values (as specified by the variable range and the nominal value). In this case, ADE GXL cannot determine a higher value (if the current value is at the upper boundary) or lower value (if at lower boundary) for computing its sensitivity. In this case, ADE GXL only uses two points for calculating the sensitivity and the correlation will always be 1 or -1, since two points will always form a straight line. However, the regression values (for this variable) are still applicable (since they represent the slope of the line).

For boundary conditions where there are only two points available for simulation, the correlation is +1 or -1, depending upon whether the regression coefficient is positive or negative. However, it does not reflect the true meaning of correlation because you need to specify at least three points for calculating the correlation coefficients.

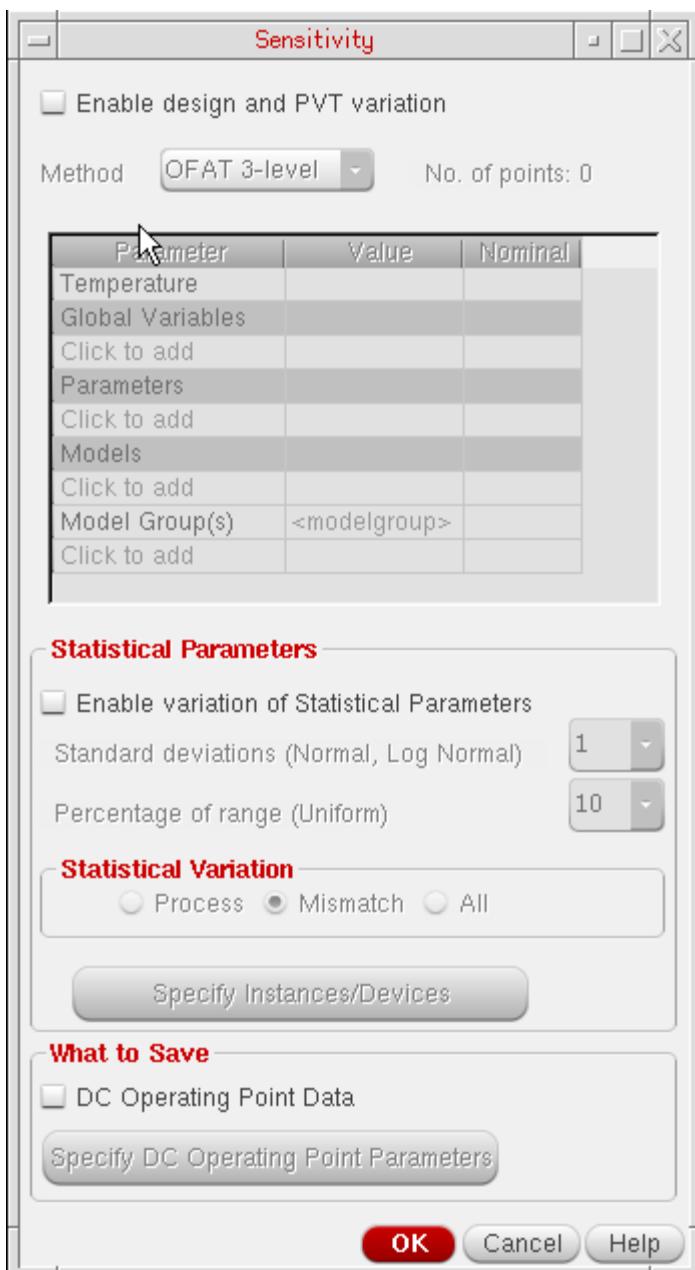
To generate and view the sensitivity data:

1. Select *Sensitivity Analysis* in the *Select a Run Mode* drop-down list on the Run toolbar, then click the *Simulation Options*  button on the Run toolbar.

The Sensitivity form appears, as shown below.

# Virtuoso Analog Design Environment GXL User Guide

## Sensitivity Analysis



**Note:** The *Edit Reference Point* command on toolbar is disabled when you select *Sensitivity Analysis* in the *Select a Run Mode* drop-down list.

2. Select the *Enable Design and PVT Variation* check box if you want to vary temperature, global variables, device parameters, or model files for sensitivity analysis.

Ensure that if you select this check box, you have specified at least one variable or parameter in the Variables and Parameters assistant. The variables or parameters that

you have included in this form appear with a strikethrough in the Variables and Parameters assistant and the Data View pane.

Also ensure that you disable all other variables or parameters that have sweep values in the Variables and Parameters assistant and are not overridden in the Sensitivity form. This is because Sensitivity Analysis does not consider variables with sweep values in the Variables and Parameters assistant and displays an error.

Then, do the following:

- a. In the *Method* field, specify the method to be used to vary global variable and parameter values. You can select any one of the following two values:

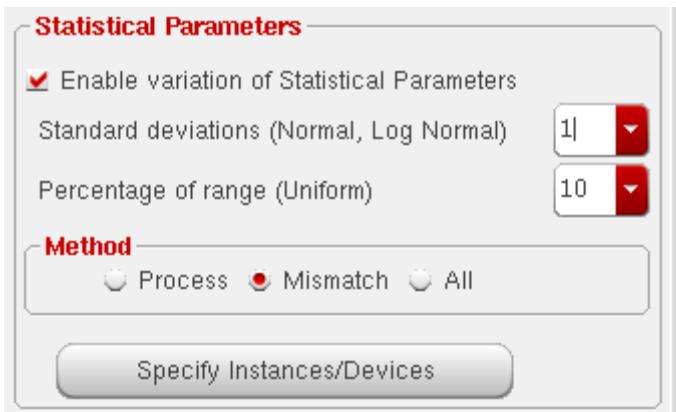
- OFAT 3-level
- OFAT sweep

For more details on these methods, refer to [Variation of Design Variables and Device Parameters](#).

- b. In the table given below the *Method* field, select the design variables, parameter, or model files that you want to vary. For more details, refer to [Specifying Variables, Parameters, and Model Files to be Varied for Sensitivity Analysis](#) on page 28.
- c. Depending on the range of values and nominal value and the value in the *Method* list, the tool calculates the total number of data points for the sensitivity analysis and displays the count in the *No. of Points* label to the right of the *Method* field, as shown below:

Parameter	Value	Nominal
Global Vari...		
IREF	45u,50u,55u	50u
Click to add		
Parameters		
M3/fw	1u:0.1u:5u	3.3u
M1/fw	1u:0.1u:10u	3.7u
M12/fw	1u:0.1u:7.5u	3.1u
M6/fw	1u:0.1u:10u	1.6u

3. Select the *Enable variation of Statistical Parameters* check box if you want to vary statistical process and mismatch parameters.



Then, do the following:

- a. In the *Standard deviations (Normal, Log Normal)* field, specify the number of standard deviations for statistical parameters with normal or log normal distribution.
- b. In the *Percentage of Range (Uniform)* field, specify the percentage range by which statistical parameters with uniform distribution need to be varied.

**Note:** The value specified must be a number greater than 0 and no more than 50.

- c. In the *Method* group box, select one of the following statistical variations:

*Process* for process statistical variations

*Mismatch* for per-instance statistical variations

*All* for both process and per-instance statistical variations

- d. (Optional) In the step c above, if you select *Mismatch* or *All*, by default, the tool varies mismatch statistical parameters for all devices and instances in the design. Click *Specify Instances/Devices* to select specific instances and devices for which you want to analyze the impact of statistical variations. For more details, refer to [Selecting Instances and Parameters for Mismatch Variation](#) on page 33.
4. (Optional) By default, you cannot view the sensitivity data for DC operating point parameters because the data is not saved in the results database. To save and view sensitivity data for specific DC operating point parameters, do the following:
  - a. Select the *DC Operating Point Data* check box.
  - b. Click the *Specify DC Operating Point Parameters* button to specify the DC operating point parameters for which you want to save and view sensitivity data. For

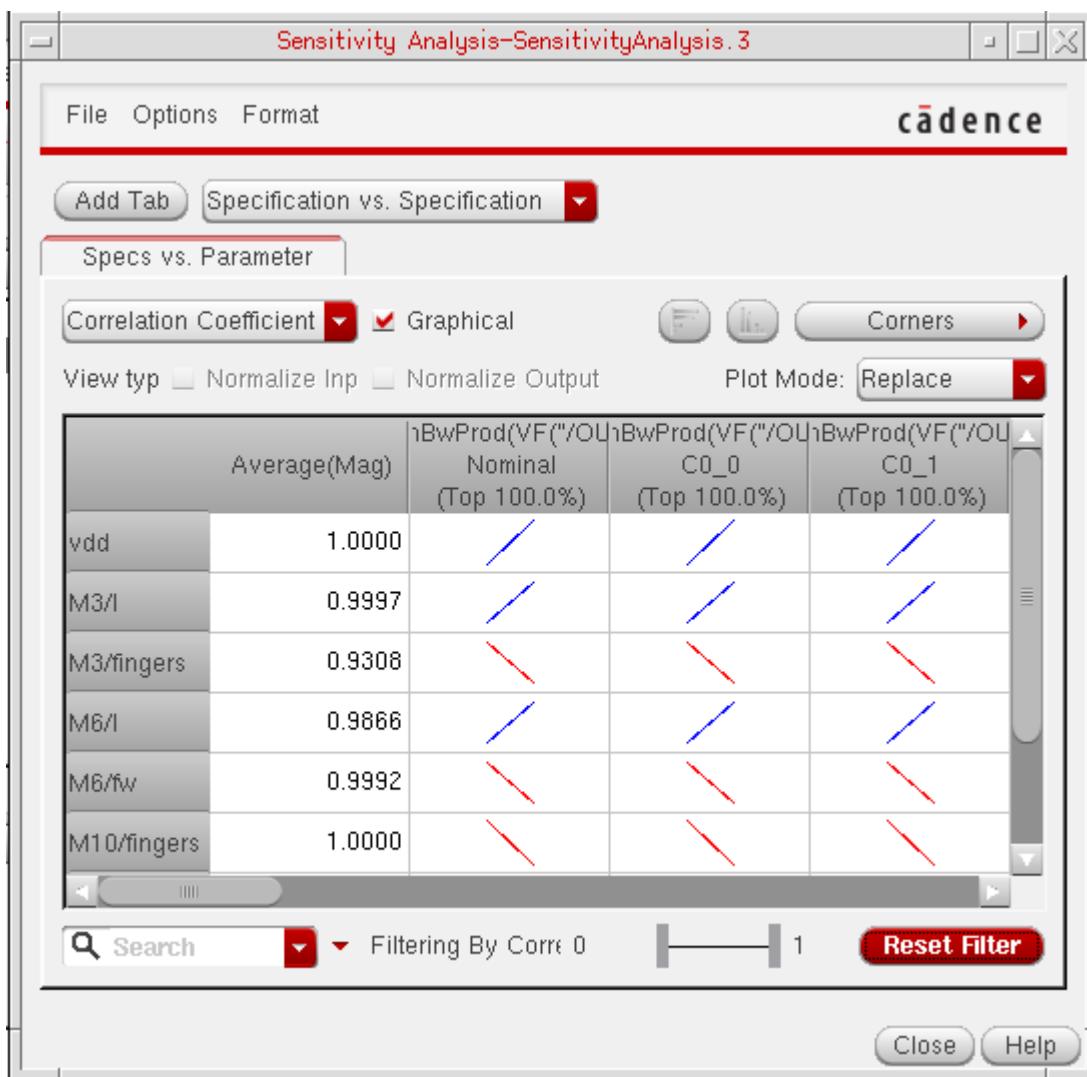
## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

more information, see [Saving Sensitivity Data for DC Operating Point Parameters](#) on page 40.

5. Click *OK* to save the changes and close the Sensitivity form.
6. Click the *Run Simulation*  button on the Run toolbar.

After the run is complete, ADE GXL displays the simulation results on the Results tab. In addition, it also opens the **Sensitivity Analysis** window that displays the sensitivity data for different specs or parameters. By default, this window displays the correlation coefficient results in graphical format, as shown below.

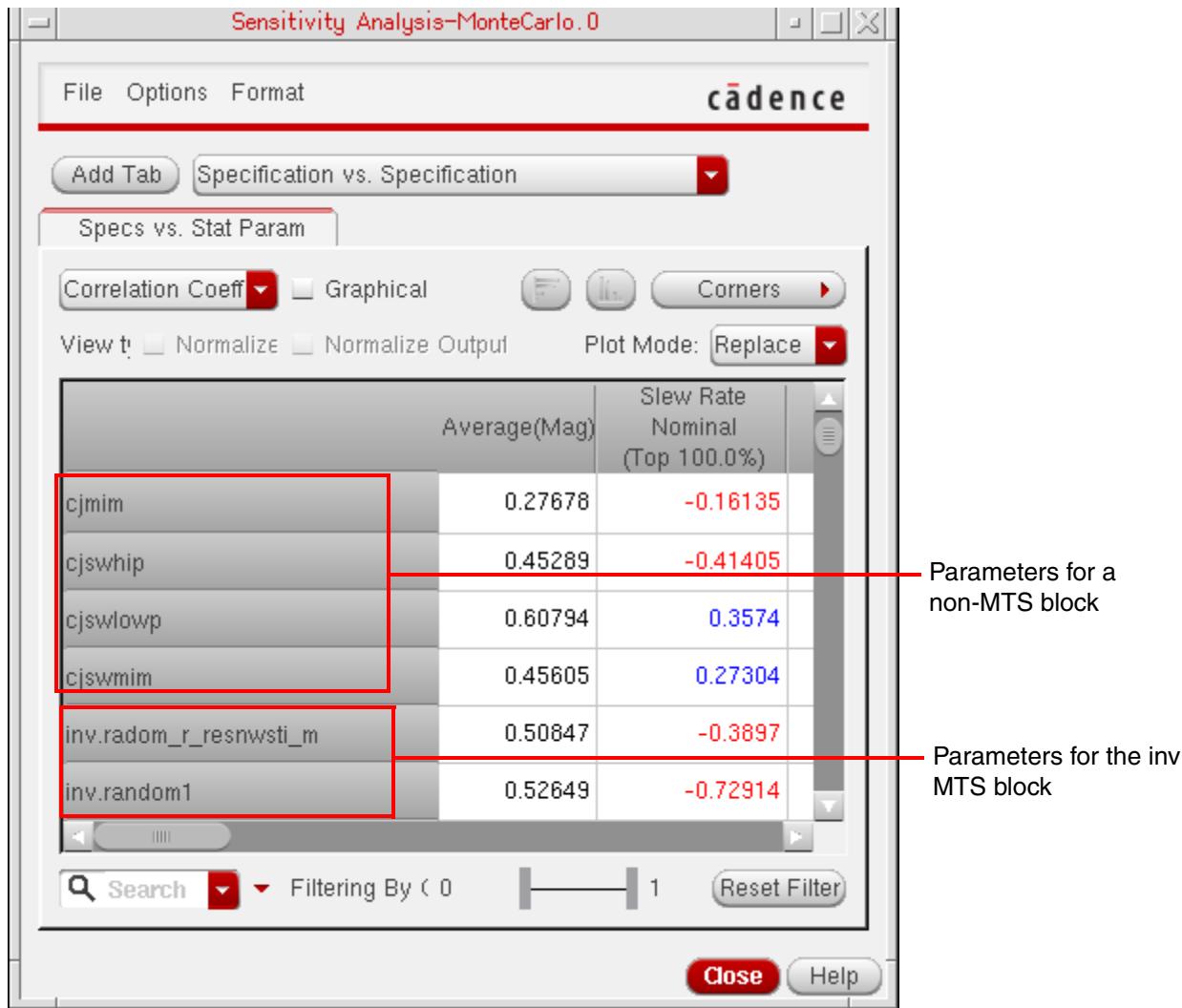


You can view the data in other formats. For more details, refer to [Working with Sensitivity Data](#).

## Support for Multi-Technology Simulations in Sensitivity Analysis

If you have enabled multi-technology simulations for a test, in the Sensitivity Analysis results window, the parameter names for MTS blocks are prefixed with the name of their corresponding block.

For example, in the sensitivity analysis results displayed below, the names of parameters for the `inv` MTS block are prefixed with `inv`.



## Troubleshooting a Design Point from the Sensitivity Results

To troubleshoot a point from the sensitivity results, in the Results tab, right-click in the data cell and choose *Troubleshoot Point*.

**Note:** If you troubleshoot a design point from existing sensitivity analysis results, the Sensitivity Analysis window is not displayed after the run. This is because sensitivity analysis cannot be performed on the results of a troubleshoot point. The following commands are also disabled for the sensitivity analysis results of a troubleshoot point:

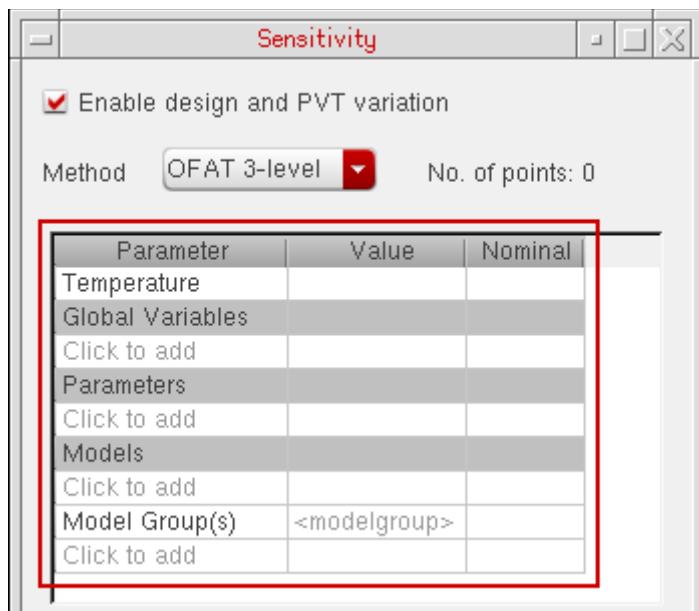
- *Sensitivity Analysis* command in the shortcut menu of a history item on the *History* tab
-  button on the toolbar in the *Results* tab

See the following topics for more information about using the Sensitivity form:

- [Specifying Variables, Parameters, and Model Files to be Varied for Sensitivity Analysis](#) on page 28
- [Selecting Instances and Parameters for Mismatch Variation](#) on page 33
- [Viewing Sensitivity Data](#) on page 54
- [Working with Sensitivity Data](#) on page 60
- [Plotting Sensitivity Analysis Results](#) on page 80
- [Highlighting Associated Devices in the Schematic](#) on page 77
- [Saving the Sensitivity Data](#) on page 95

## Specifying Variables, Parameters, and Model Files to be Varied for Sensitivity Analysis

You can specify the design variables, parameters, or model files that you want to vary for sensitivity analysis in the table given on the Sensitivity form.



In this table, you can select the name of a variable and specify values for which you want to vary the variable and a nominal value for it.

To vary temperature:

1. Double-click in the *Value* cell for *Temperature* and specify a range or a set of values for which you want to vary temperature.
2. Double-click in the *Nominal* cell for temperature and open the drop-down list.  
All the values specified in the *Value* cell are displayed in the list.
3. Select a nominal value from this list.

Alternatively, you can get the value of temperature from the test or a reference point. For this, right-click in the *Nominal* cell for *Temperature* and choose an appropriate command from the following options:

- Get Value from Test*: Copies the value of temperature from the test.
- Get Value from Reference Point*: Copies the value of temperature from the reference point created using the Edit Reference Point form.

**Note:** If the value taken from the test or reference point does not exist in the *Value* cell, the tool automatically adds the nominal value to the existing values. For example, if you specify -40 and 85 as two values for temperature and get the nominal value from test, which is 27, the tool adds 27 to the value list.

Parameter	Value	Nominal
Temperature	-40 27 85	27

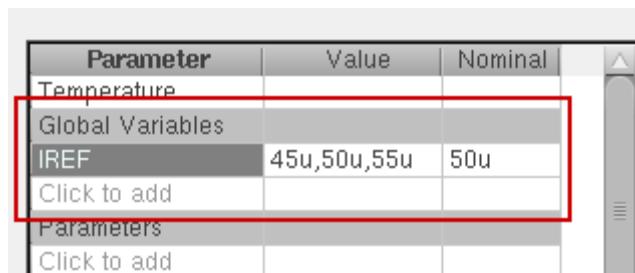
To select global variables to be varied:

1. Click on *Click to add* in the Global Variables section in the table.

A list of global variables given in the Data View pane is displayed in the cell.

2. From the list, select a global variable that you want to vary.

The tool gets the sweep values and the nominal value specified for the variable in the Variables and Parameters Assistant and displays it in the *Value* and *Nominal* cells, respectively, as shown below.



Parameter	Value	Nominal
Temperature		
Global Variables		
IREF	45u,50u,55u	50u
Click to add		
Parameters		
Click to add		

If required, you can change the range or nominal value for a variable in this table. For more details, refer to [Changing the Parameter Value](#) and [Changing the Nominal Value](#).

To select design parameters to be varied:

1. Click on *Click to add* in the Parameters section in the table.

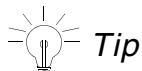
A list of parameters already defined in the Variables and Parameters Assistant is displayed in the cell.

2. From the list, select the parameter that you want to vary.

The tool reads the range of values and design value for the selected parameter from the Variables and Parameters Assistant and displays it in the *Value* and the *Nominal* column, respectively, as shown below.

Parameter	Value	Nominal
Global Variations		
IREF	45u,50u,55u	50u
Click to add		
Parameters		
M3/fw	1u:0.1u:5u	3.3u
M1/fw	1u:0.1u:10u	3.7u
M12/fw	1u:0.1u:7.5u	3.1u
M6/fw	1u:0.1u:10u	1.6u
M3/I	150n:50n:1u	550n
M1/I	150n:50n:2u	1.4u

If required, you can change the range or nominal value for a variable in this table. For more details, refer to [Changing the Parameter Value](#) and [Changing the Nominal Value](#).



*Tip*

You can move the mouse over the row of a parameter to view the library/cell/view information for the device parameter.

To select model files to be varied:

1. Click on *Click to add* in the Models section in the table.  
The Add/Edit Model Files form is displayed.
2. Browse and select a model file for which you want to vary different sections.  
The name of the file appears in the cell.
3. Double-click in the *Value* cell.  
A list of sections defined in the specified model file appears.
4. Select a section that you want to add to the list.  
You can select multiple sections that you want to vary for the model file.

5. Double-click in the *Nominal* cell and select name of a section that you want to use as a nominal value for the selected model file, as shown below.

Parameter	Value	Nominal
M3/l	150n:50n:1u	550n
M1/fw	1u:0.1u:10u	3.7u
M1/l	150n 2u	1.4u
M12/l	150n 2u	1.7u
M12/fw	1u 1.1u 1.2u ...	1.3u
Click to add		
Models		
gpdk045.scs	<input checked="" type="checkbox"/> ss tt ff	tt
Click to add		
Model Group(s)	<modelgroup>	
Click to add		

Similarly, you can select model groups that you want to vary for the simulation.

 **Important**

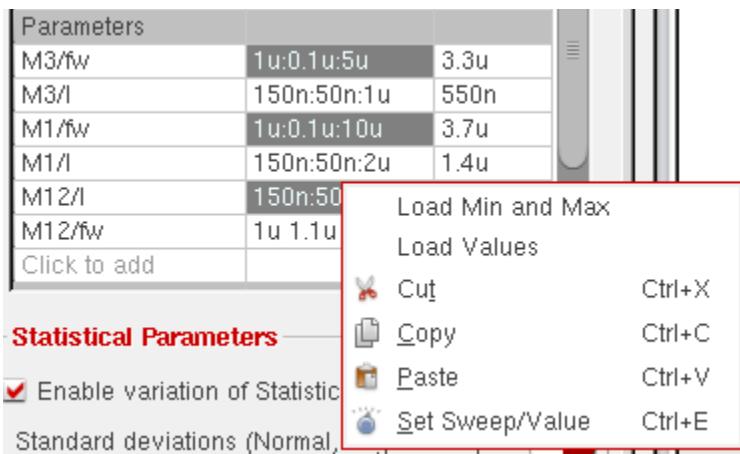
You must define your models so that they respond to the statistical variations you choose. For a Spectre circuit simulator example on how to define your models, see Specifying Parameter Distributions Using Statistics Blocks in the *Virtuoso Spectre Circuit Simulator User Guide*.

### Changing the Parameter Value

To change the values for a parameter, double-click in the *Value* cell for the variable or parameter and edit the value. You can either specify a range or specify a list of space-separated values, as shown below.

M1/fw	1u:0.1u:10u	3.7u
M1/l	150n:50n:2u	1.4u
M12/l	150n:50n:2u	1.7u
M12/fw	1u 1.1u 1.2u 1.3u 1.4u 1.5...	1.3u

Alternatively, you can select one or more values and right-click to display the shortcut menu:



From this menu, choose:

- *Load Values* to load the values defined in the Data View or the Variables and Parameters Assistant
- *Load Min and Max* to load only the minimum and maximum values defined in the Data View or the Variables and Parameters Assistant.
- *Set Sweep/Value* to edit the exclusion/inclusion list.

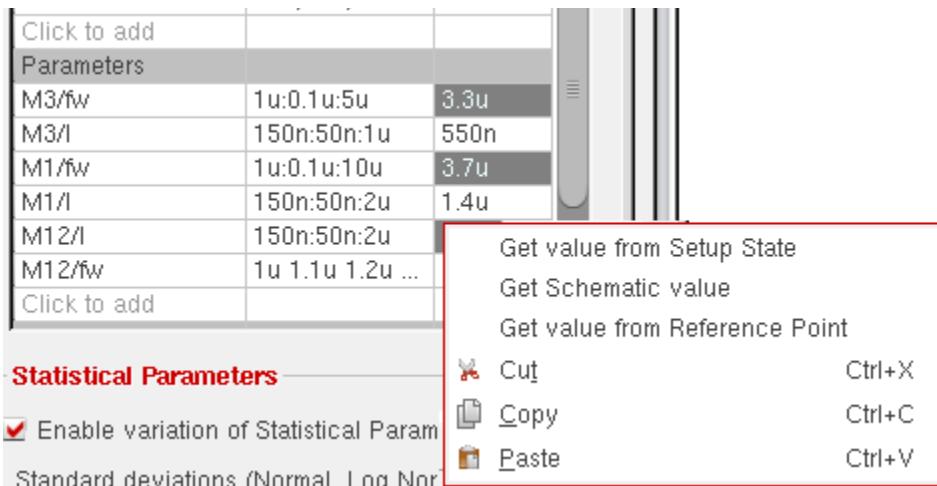
### Changing the Nominal Value

The default nominal value is taken from the values of the parameter in the design. To change the nominal value for a variable or parameter, double-click in the *Nominal* cell for the variable or parameter. A list of possible nominal values is displayed. You can either select a value from the list or edit the value. Note that:

- If the variation method is OFAT 3-level, a list of all possible nominal values calculated using the range of values is displayed.
- If the variation method is OFAT Sweep, all sweep values appear in the list. You can select any value from this or type a different value.

**Note:** If you specify a nominal value outside the specified range or the sweep set, the tool does not accept the value and makes the cell blank. A valid nominal value for a parameter is any value from within the range or sweep set corresponding to the parameter.

Alternatively, select one or more cells in the Nominal column and right-click to display the following shortcut menu:



and choose any one of the following commands:

- *Get value from Setup State* to get the nominal value from a saved setup state.

The Get Nominal Values from Setup State form is displayed. In the *State Name* list, select the name of a saved state. In the *What to Load* section, select *Parameters*. This option copies the values of all the parameters from the selected saved state.

- *Get Schematic value* to get the nominal value from the schematic.

**Note:** This command is not available for global variables. Instead, the *Get value from Test* command is available. You can use this command to use the value of variable from the test.

- *Get value from Reference Point* to get the nominal value from the reference table. For this, ensure that you have specified reference values for parameters in the Edit Reference Point form.

**Note:** If the value taken from the setup state, schematic, or reference point does not fall in the given value range or sweep set, the tool automatically expands the range or adds the schematic value to the sweep set.

## Selecting Instances and Parameters for Mismatch Variation

By default, statistical analysis analyzes mismatch variations for all devices and instances in the design. If the number of instances is large, time to run simulation is very high. You can limit mismatch variation analysis to run for a selected set of instances or devices. For this, click *Specify instances/Devices* on the Sensitivity options form.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

The tool netlists the design for the enabled tests and extracts data on the instances modeled by statistical mismatch.

**Note:** Netlist is created only when you are selecting instances for a set of tests for the first time. When you click on *Specify instances/Devices* for consecutive selections, netlist is created for only those designs/tests that were changed after the previous selections.

After the dummy simulations are run successfully, the Select Instances and Parameters for Mismatch form is displayed, as shown in the following figure.



Only those instances that are modeled with mismatch variation will be listed in the *Choices* lists.

To select instances and parameters, do the following:

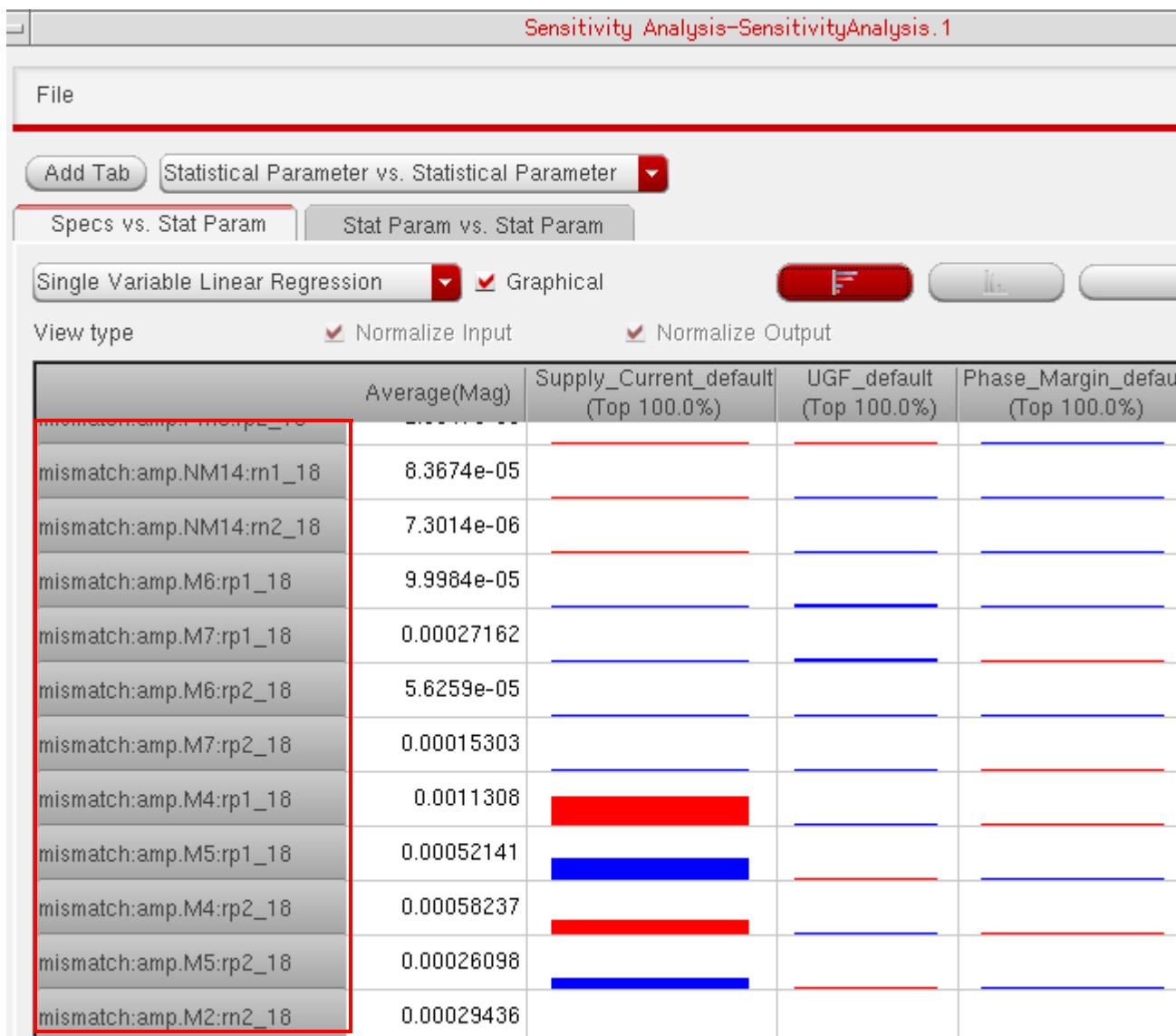
1. In the *Instance Selection Method* section, select a method by which you want to select instances. You can select instances in any of the following three ways:

- By selecting instances on schematic. For this, select the *Schematic* option in the *Instance Selection Method* section. For more details, refer to [Selecting Instances from Schematic](#) on page 36.
  - By selecting instances of master cellviews. For this, select the *Master* option in the *Instance Selection Method* section. For more details, refer to [Selecting Instances from Master Cellviews](#) on page 38.
  - By selecting subcircuit instances. For this, select the *Subcircuits* option in the *Instance Selection Method* section. For more details, refer to [Selecting Instances from Subcircuits](#) on page 39.
2. Click *OK* to save the changes and return to the Sensitivity options form.

Next time when you run sensitivity analysis, the mismatch variation is run for only the selected instances. For example, in the following figure, the variation is limited to only selected instances and parameters.

# Virtuoso Analog Design Environment GXL User Guide

## Sensitivity Analysis



### Selecting Instances from Schematic

To select instances from schematic, do the following:

1. In the *Instance Selection Method* section, select the *Schematic* option.
2. In the *Test* list, select the name of test for which you want to open the schematic view.  
**Note:** Select *All Tests* if you want to select instances for all the tests that are enabled in the *Data View* pane.
3. Click *Select Instances*. The schematic view for the selected test is opened in a new tab.

4. On the schematic tab, hold the *Shift* key and select instances that you want to include for mismatch variation.
5. Press the *Esc* key when done.

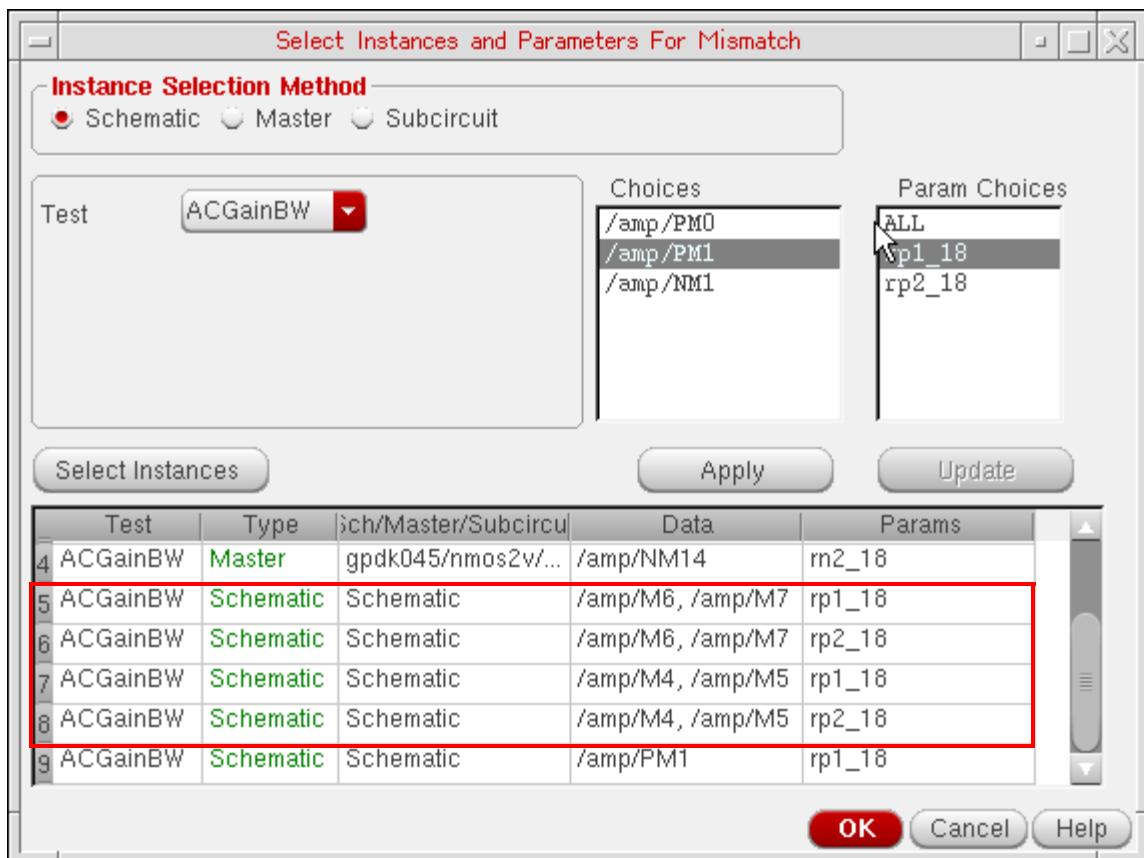
The list of selected instances that are modeled with statistical mismatch variation is displayed in the *Choices* list.

**Note:** The selected instances that are not modeled with statistical mismatch variation will not be added to the *Choices* list. Instead, an error message 1692 is displayed with the names of all such instances.

6. In the *Choices* list, select an instance.

modeled parameters corresponding to the selected instances are displayed in the *Param Choices* list.

7. In the *Param Choices* list, select ALL or specific parameters that you want to vary for mismatch variation.
8. Click *Apply* to save the selections as rows in the tabular list, as shown in the following figure.



## Selecting Instances from Master Cellviews

To select instances from cellviews, do the following:

1. In the *Instance Selection Method* section, select the *Master* option.

List of all the libraries, cells, and views are displayed for the selected test.

2. Select the library, cell, and view name for which you want to select instances.

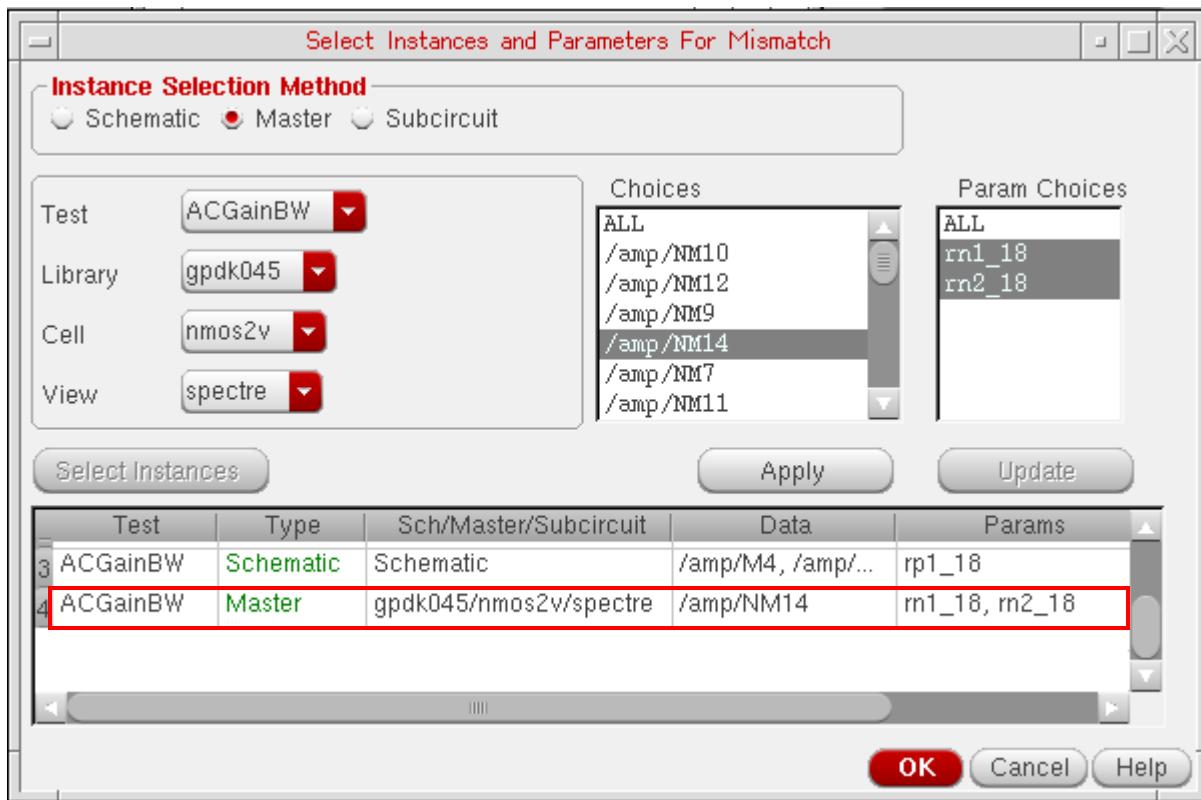
For the selected cellview, the list of instances that are modeled with statistical mismatch variation are displayed in the *Choices* list.

3. In the *Choices* list, select one or more instances.

The modeled parameters corresponding to the selected instances are displayed in the *Param Choices* list.

4. In the *Param Choices* list, select ALL or specific parameters that you want to vary.

5. Click *Apply* to save the selections as rows in the tabular list, as shown in the following figure.



## Selecting Instances from Subcircuits

To select instances from subcircuits, do the following:

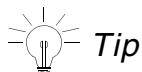
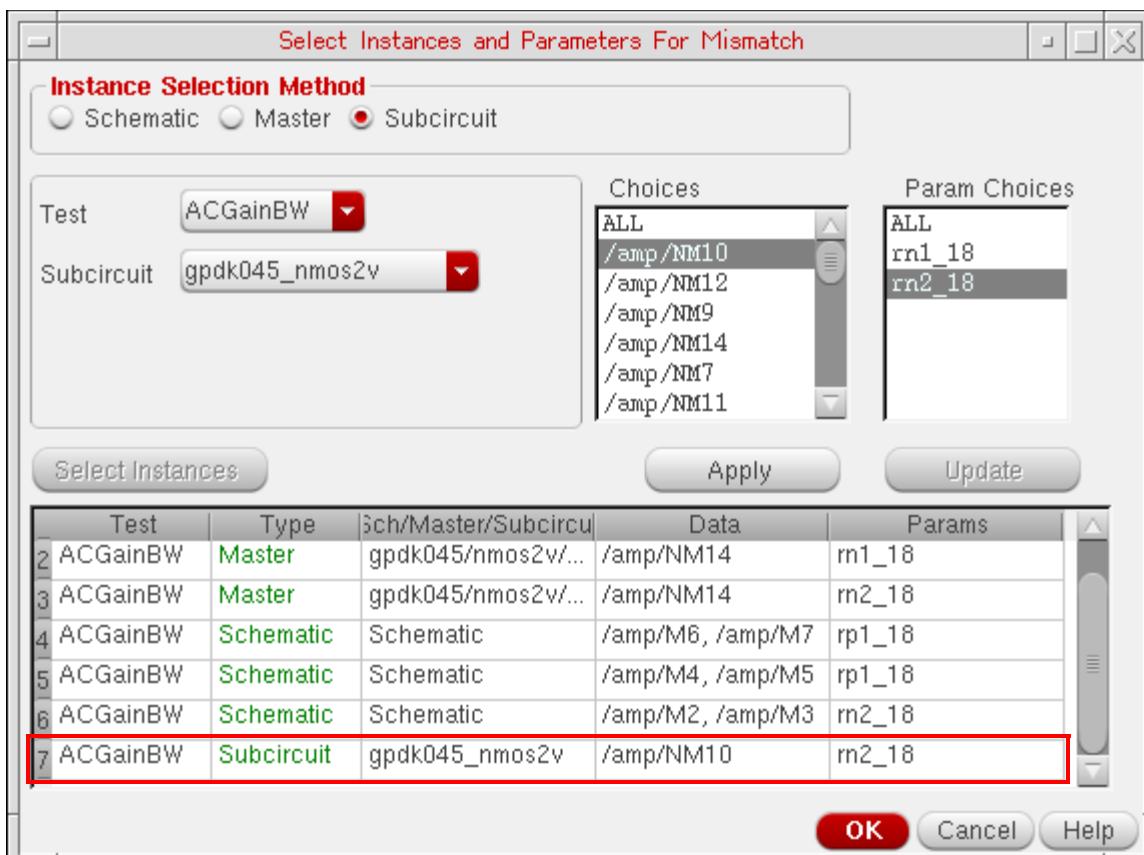
1. In the *Instance Selection Method* section, select *Schematic*.
2. In the *Test* list, select the name of test for which you want to open the schematic view.  
**Note:** Select *All Tests* if you want to select instances for all the tests that are enabled in the *Data View* pane.
3. In the *Subcircuit* list, choose the subcircuit whose instances you want to select.  
All the instances in the selected subcircuit that are modeled with statistical mismatch variation are displayed in the *Choices* list.
4. In the *Choices* list, select one or more instances.

The modeled parameters corresponding to the selected instances are displayed in the *Param Choices* list.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

5. In the *Param Choices* list, select ALL or specific parameters that you want to vary.
6. Click *Apply* to save the selections as rows in the tabular list, as shown in the following figure.



*Tip*  
To delete any selection from the tabular list, select the required row(s), right-click and choose *Delete*.

## Saving Sensitivity Data for DC Operating Point Parameters

By default, you cannot view the sensitivity data for DC operating point parameters because the data is not saved in the results database. The Select OP Point Parameters form allows you to specify the DC operating point parameters for which you want to save and view sensitivity data. For more information about viewing sensitivity data, see [Viewing Sensitivity Data on page 54](#).

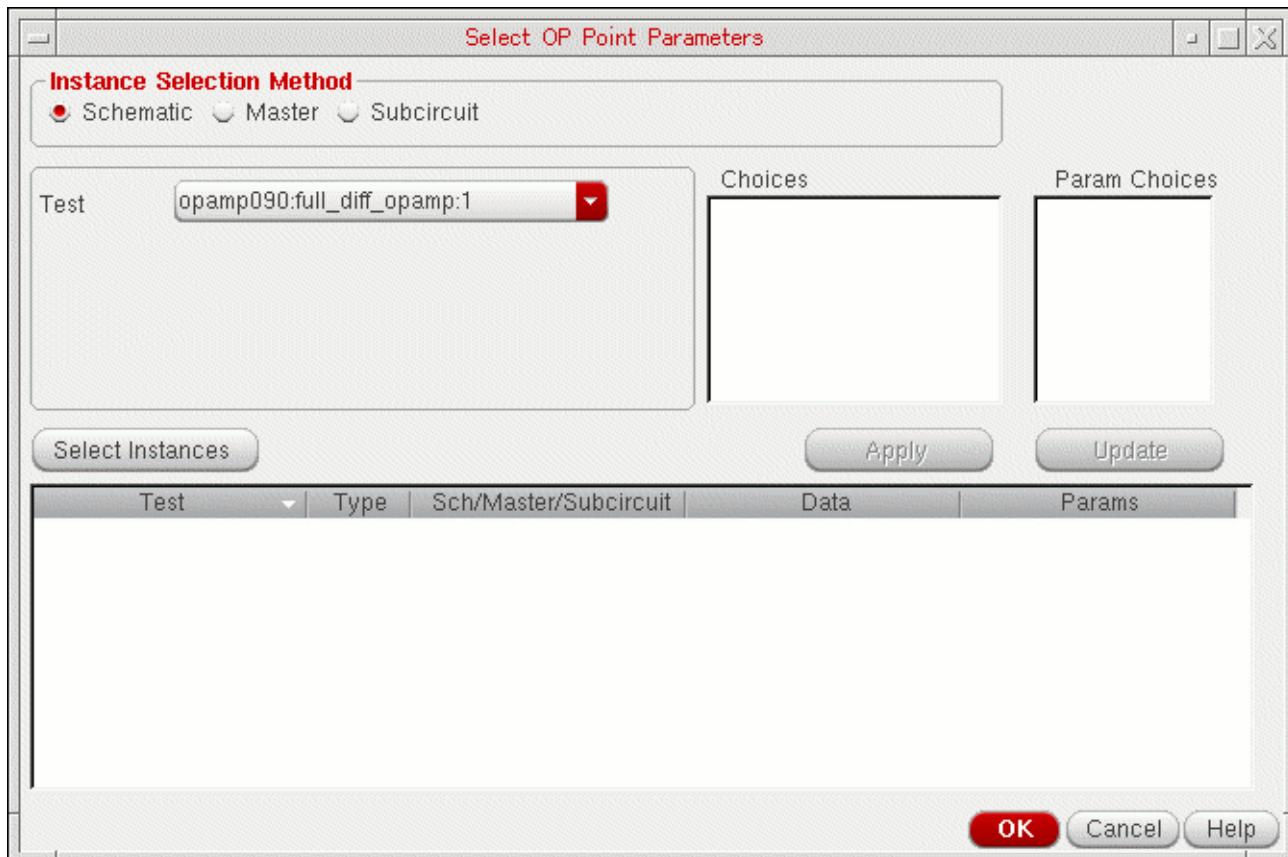
## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

To specify the DC operating point parameters for which you view sensitivity data, do the following:

1. On the *Sensitivity* form click the *Specify DC Operating Point Parameters* button.

The Select OP Point Parameters form appears.



2. Do one of the following:

Select	To
<i>Schematic</i>	Add DC operating point parameters by selecting instances on the schematic.  For more information, see <a href="#">Adding DC Operating Point Parameters of Schematic Instances</a> on page 42.
<i>Master</i>	Add DC operating point parameters of instances of cellviews.  For more information, see <a href="#">Adding DC Operating Point Parameters of Cellview Instances</a> on page 46.

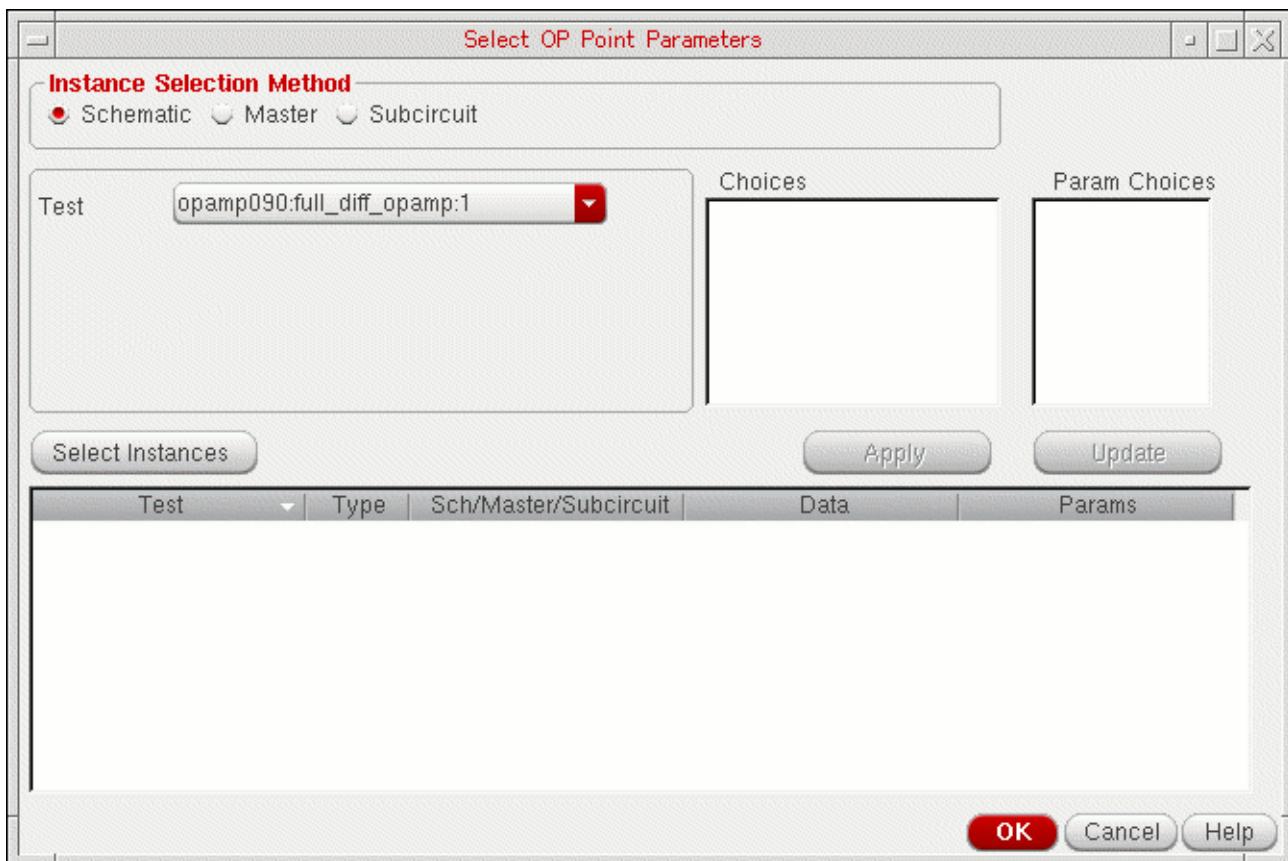
Select	To
<i>Subcircuit</i>	Add DC operating point parameters of subcircuit instances. For more information, see <a href="#">Adding DC Operating Point Parameters of Subcircuit Instances</a> on page 50.

3. Click *OK* to save the changes and return to the Sensitivity options form.

### **Adding DC Operating Point Parameters of Schematic Instances**

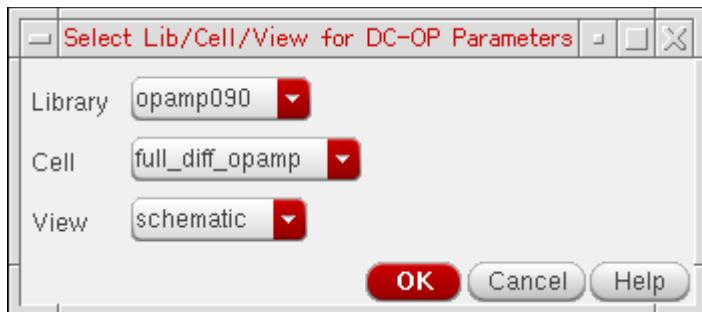
To add DC operating point parameters by selecting instances on the schematic, do the following:

1. Select the *Schematic* option.



2. In the *Test* drop-down list, choose the test in which you want to select instances.

To select instances for all the tests that are enabled in the Data View pane, choose *All Tests* in the *Test* drop-down list. If you choose *All Tests*, the Select Lib/Cell/View for DC-OP Parameters form appears.



Do the following to open the schematic view from which you want to select instances for all the tests that are enabled in the Data View pane.

- a. Use the *Library*, *Cell* and *View* drop-down lists to select the schematic view.

**Note:** The *Library*, *Cell* and *View* drop-down lists display only the cellviews that are used in the designs for all the tests.

- b. Click *OK*.

The selected schematic view is opened in a new tab.

3. If a test name is selected in the *Test* drop-down list, click the *Select Instances* button. The schematic for the test is opened in a new tab.
4. In the schematic, select one or more instances.

To select more than one instance at a time, do one of the following:

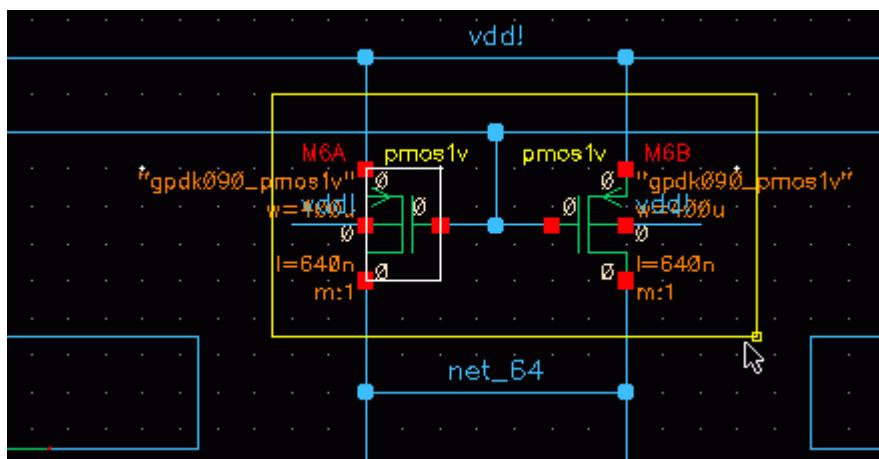
- Hold down the *Shift* key and click on instances.
- Click and drag the mouse over the instances you want to select.

All the instances that are within the yellow bounding box that appears are included in the selection.

## Virtuoso Analog Design Environment GXL User Guide

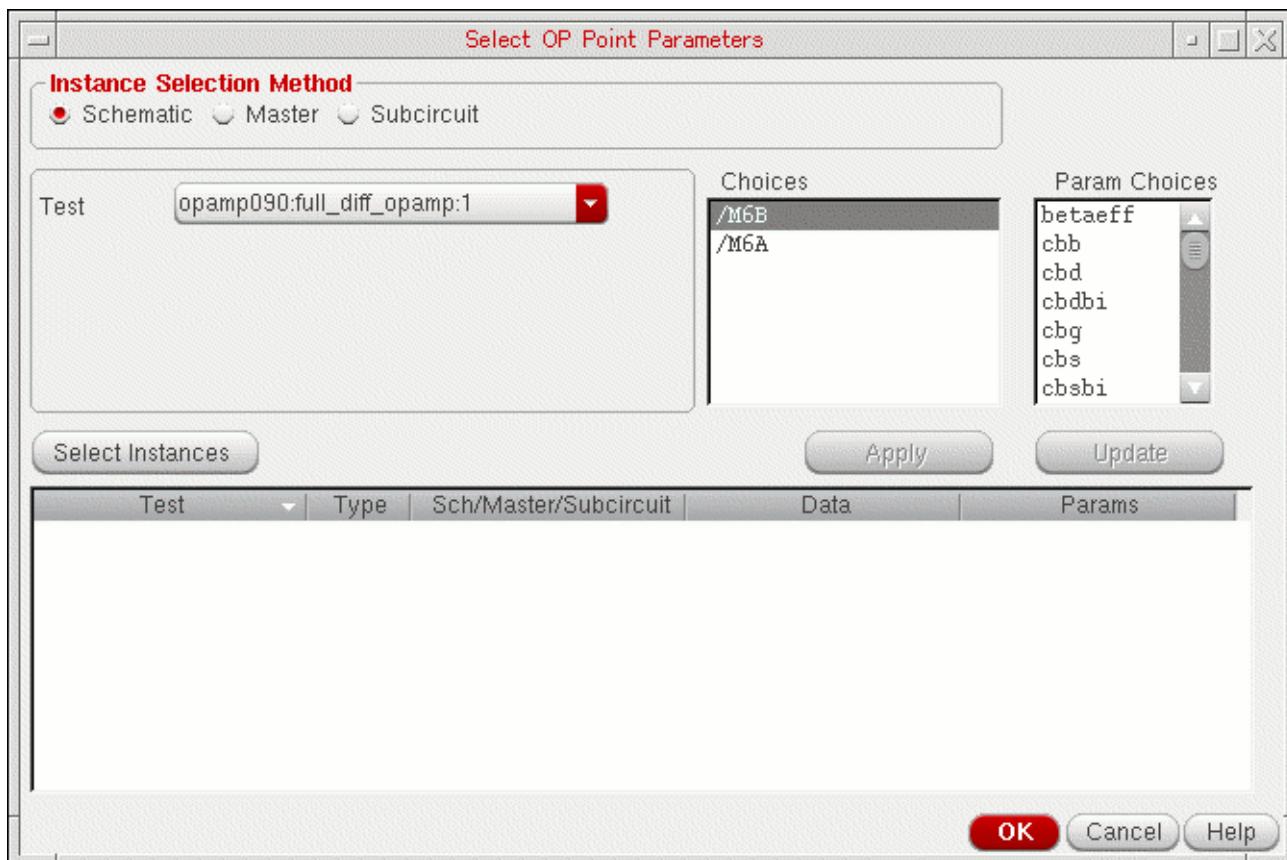
### Sensitivity Analysis

In the following example, instances M6A and M6B that are within the yellow bounding box are included in the selection.



5. Press the *Esc* key when you are done.

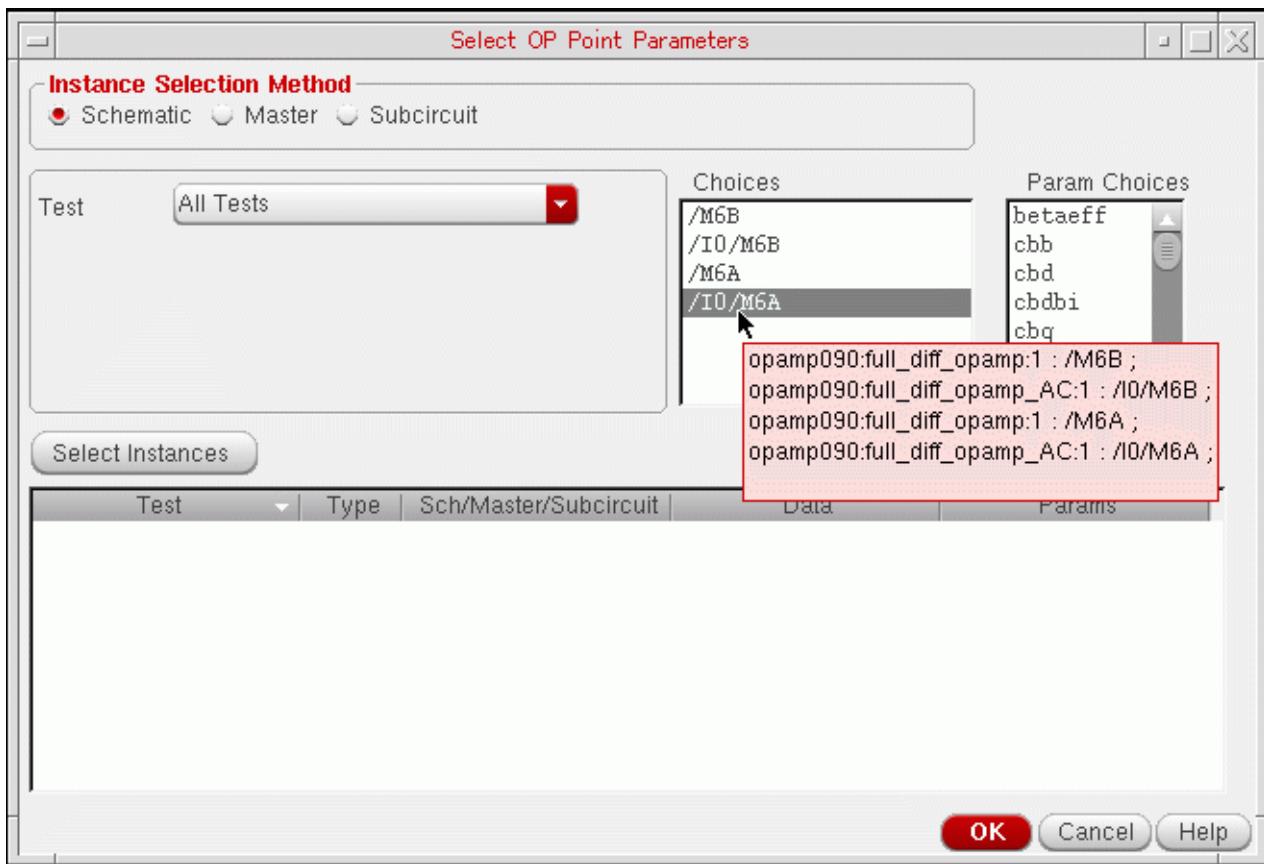
The selected instances are displayed in the *Choices* field.



## Virtuoso Analog Design Environment GXL User Guide

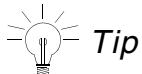
### Sensitivity Analysis

**Note:** If *All Tests* is selected in the *Test* drop-down list, the instance names of the selected instances in the designs for all the tests are displayed in the *Choices* field. You can place the mouse pointer in the *Choices* field to view the instance names of the selected instances in the design for each test.



6. In the *Choices* field, select the instance for which you want to add DC operating point parameters.

The DC operating point parameters for the instance are displayed in the *Param Choices* field.



If the same DC operating point parameter exists in more than one instance, you can simultaneously add the parameter for all the instances by selecting those instances in the *Choices* field. To select multiple instances, hold down the *Shift* key (for contiguous selection) or the *Ctrl* key (for noncontiguous selection) and click the next instance to add it to the selection set.

7. In the *Param Choices* field, select the DC operating point parameter you want to add.

To select multiple DC operating point parameters, hold down the *Shift* key (for contiguous selection) or the *Ctrl* key (for noncontiguous selection) and click the next parameter to add it to the selection set.

**8. Click *Apply*.**

The selected parameters are added for the test. For example, in the following figure, the DC operating point parameters `betaeff` and `cbb` of instance `M6B` are added for the `opamp090:full_diff_opamp:1` test.



See also:

- [Replacing DC Operating Point Parameters](#) on page 53
- [Deleting DC Operating Point Parameters](#) on page 53

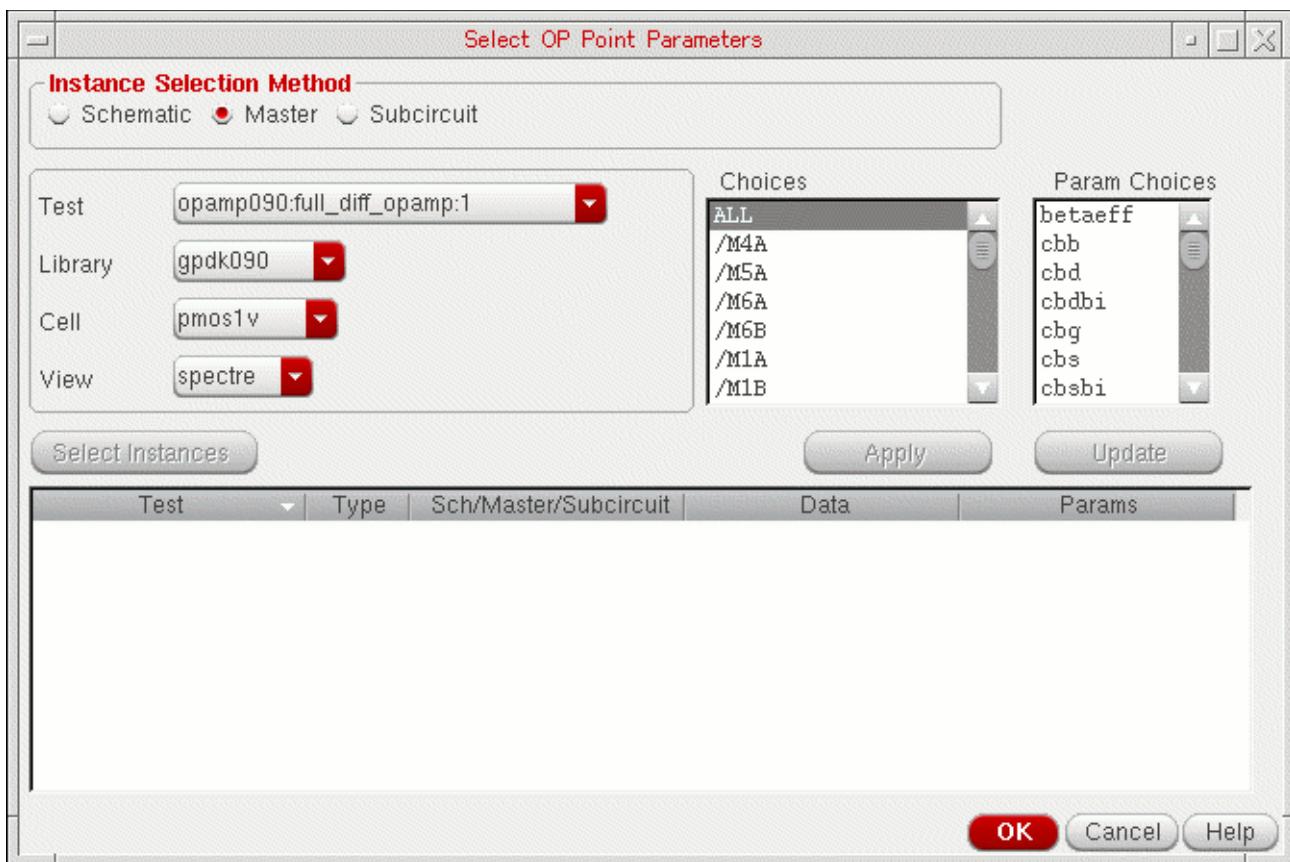
### **Adding DC Operating Point Parameters of Cellview Instances**

To add DC operating point parameters of instances of cellviews, do the following:

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

1. Select the *Master* option.



2. In the *Test* drop-down list, choose the test for which you want to select instances of cellviews.

To select instances for all the tests that are enabled in the Data View pane, choose *All Tests* in the *Test* drop-down list.

3. Use the *Library*, *Cell* and *View* drop-down lists to select the library, cell and view in which the cellview exists.

All the instances of the cellview in the design for the test are displayed in the *Choices* field.

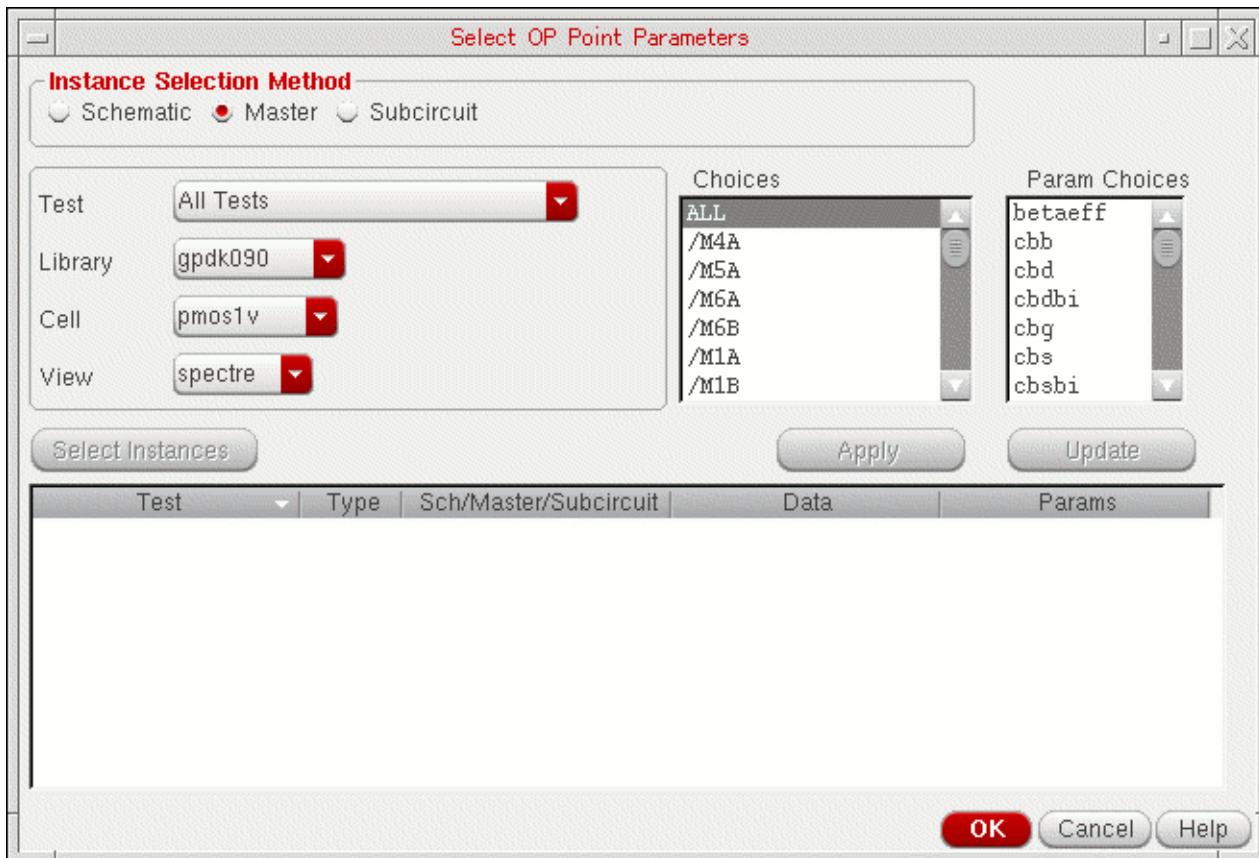
Note the following:

- ❑ The *Library*, *Cell* and *View* drop-down lists display only the libraries and cells for the cellviews that have DC operating point parameters.
- ❑ If *All Tests* is selected in the *Tests* drop-down list, the *Choices* field displays the instance names of the cellview in the designs for all the tests that are enabled in the Data View pane. For example, in the following figure, the *Choices* field displays the

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

instance names of the `spectre` cellview of the `pmos1v` cell in the designs for all the tests.



**Note:** You can place the mouse pointer in the *Choices* field to view the instance name of the cellview in the design for each test.

4. In the *Choices* field, select the instance for which you want to add DC operating point parameters.

The DC operating point parameters for the instance are displayed in the *Param Choices* field.

Note the following:

- ❑ To add DC operating point parameters for all instances of the cellview, select *ALL* in the *Choices* field.
- ❑ If the same DC operating point parameter exists in more than one instance, you can simultaneously add the parameter for all the instances by selecting those instances in the *Choices* field. To select multiple instances, hold down the *Shift* key (for

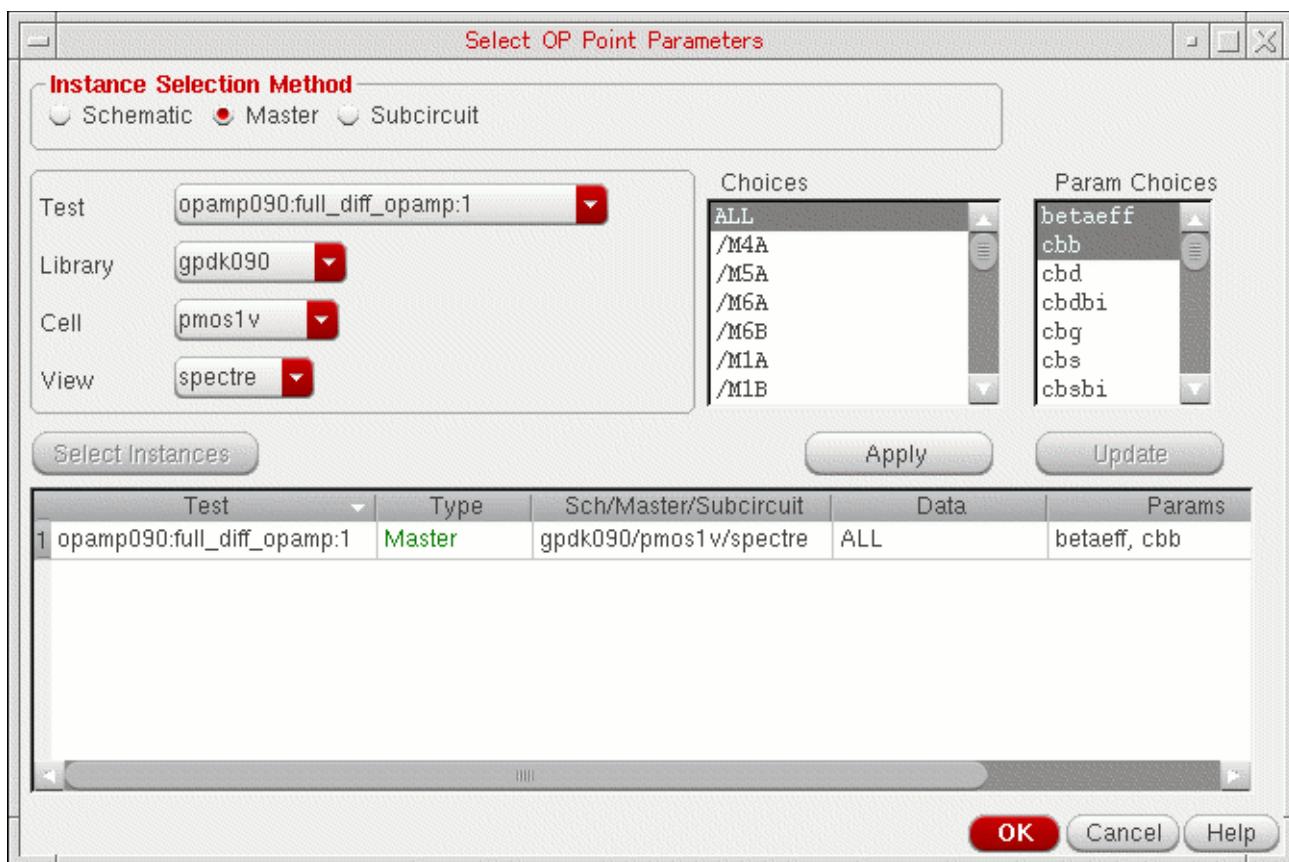
contiguous selection) or the *Ctrl* key (for noncontiguous selection) and click the next instance to add it to the selection set.

5. In the *Param Choices* field, select the DC operating point parameter you want to add.

To select multiple DC operating point parameters, hold down the *Shift* key (for contiguous selection) or the *Ctrl* key (for noncontiguous selection) and click the next parameter to add it to the selection set.

6. Click *Apply*.

The selected parameters are added for the test. For example, in the following figure, the DC operating point parameters `betaeff` and `cbb` on all instances of the `spectre` cellview of the `pmos1v` cell are added for the `opamp090:full_diff_opamp:1` test.



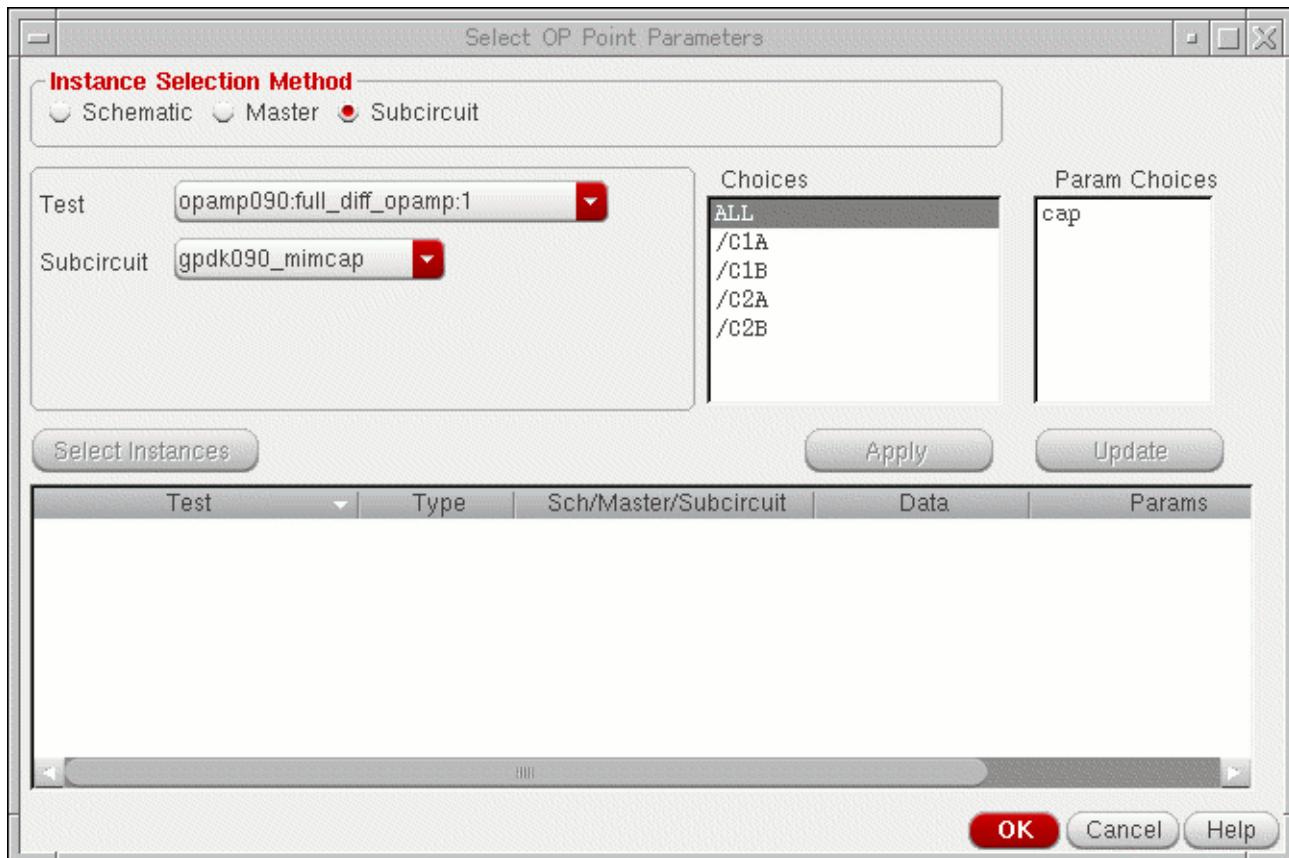
See also:

- [Replacing DC Operating Point Parameters](#) on page 53
- [Deleting DC Operating Point Parameters](#) on page 53

## Adding DC Operating Point Parameters of Subcircuit Instances

To add DC operating point parameters of subcircuit instances, do the following:

1. Select the *Subcircuit* option.



2. In the *Test* drop-down list, choose the test for which you want to select subcircuit instances.

To select subcircuit instances for all the tests that are enabled in the Data View pane, choose *All Tests* in the *Test* drop-down list.

3. In the *Subcircuit* drop-down list, choose the subcircuit whose instances you want to select.

All the instances of the subcircuit in the design are displayed in the *Choices* field.

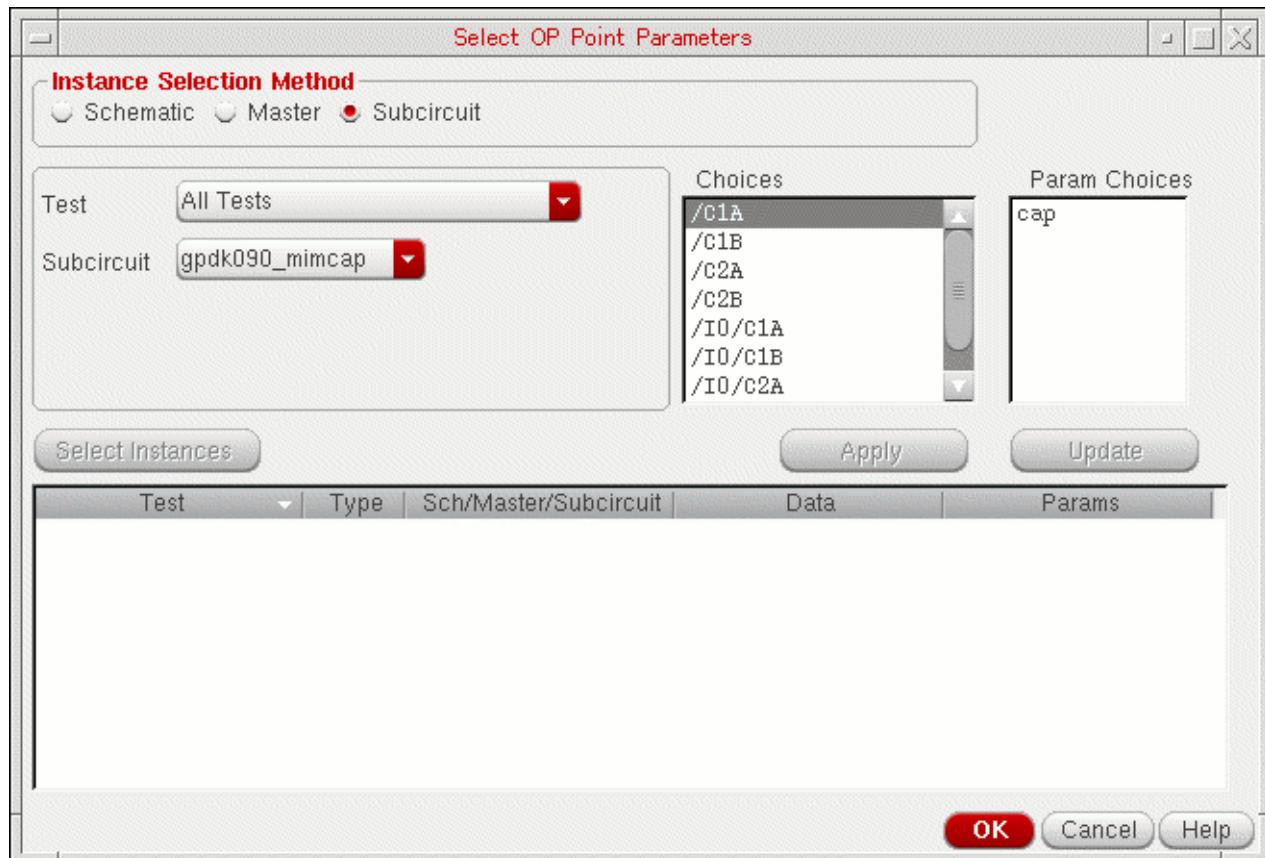
Note the following:

- ❑ The *Subcircuit* drop-down list displays only the subcircuits that have DC operating point parameters.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

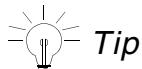
- If *All Tests* is selected in the *Tests* drop-down list, the *Choices* field displays the instance names of the subcircuit in the designs for all the tests. For example, in the following figure, the *Choices* field displays the instance names of the `ampn` subcircuit in the designs for all the tests.



You can place the mouse pointer in the *Choices* field to view the instance name of the subcircuit in the design for each test.

- In the *Choices* field, select the instance for which you want to add DC operating point parameters.

The DC operating point parameters for the instance are displayed in the *Param Choices* field.



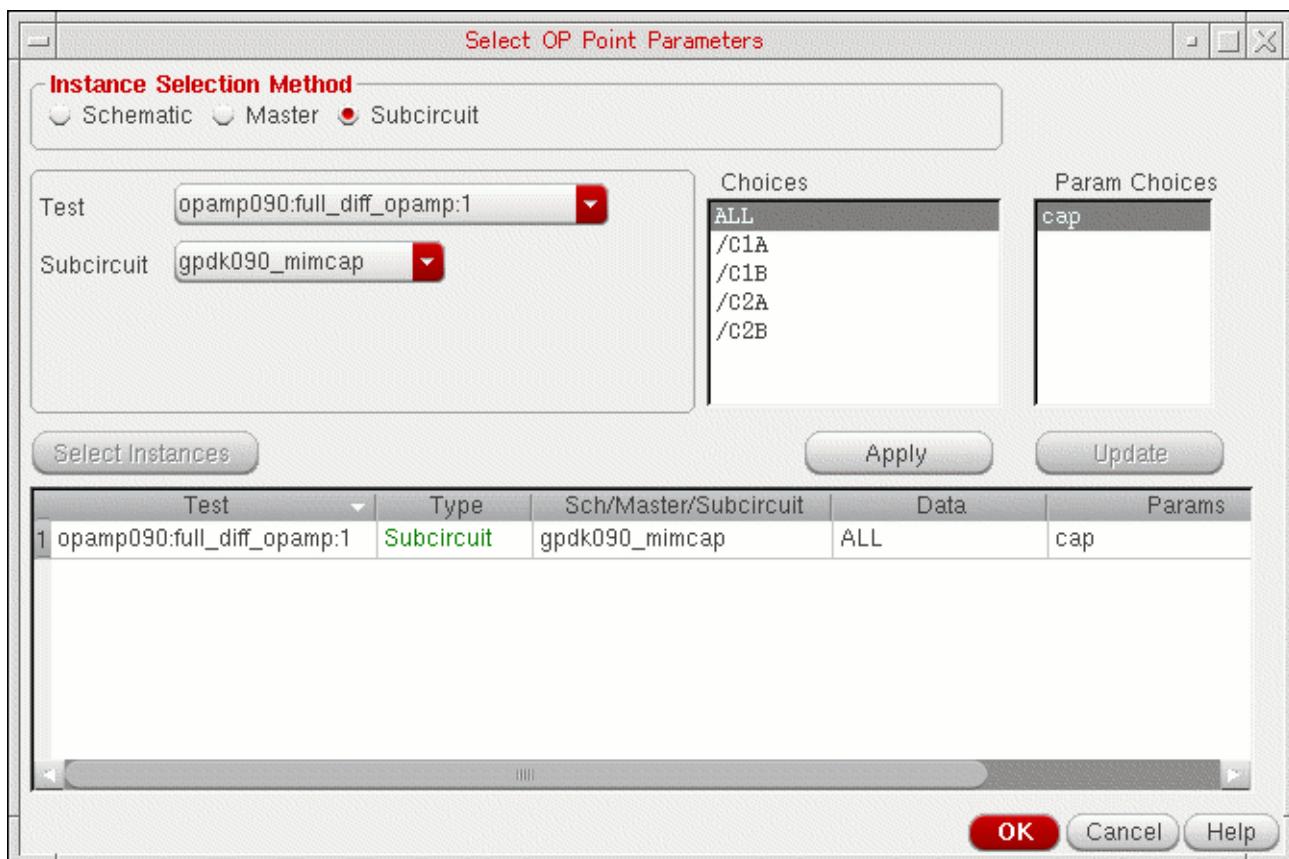
If the same DC operating point parameter exists in more than one instance, you can simultaneously add the parameter for all the instances by selecting those instances in the *Choices* field. To select multiple instances, hold down the *Shift* key (for contiguous selection) or the *Ctrl* key (for noncontiguous selection) and click the next instance to add it to the selection set.

5. In the *Param Choices* field, select the DC operating point parameter you want to add.

To select multiple DC operating point parameters, hold down the *Shift* key (for contiguous selection) or the *Ctrl* key (for noncontiguous selection) and click the next parameter to add it to the selection set.

6. Click *Apply*.

The selected parameters are added for the test. For example, in the following figure, the DC operating point parameter `cap` on all instances of the `gpdk090_mimcap` subcircuit are added for the `opamp090 : full_diff_opamp : 1` test.



See also:

- [Replacing DC Operating Point Parameters](#) on page 53
- [Deleting DC Operating Point Parameters](#) on page 53

### **Replacing DC Operating Point Parameters**

To replace one DC operating point parameter with another, do the following:

1. Select the row for the parameter.
2. Select a parameter in the *Param Choices* field.
3. Click the *Update* button.

The parameter in the row is replaced with the parameter selected in the *Param Choices* field.

### **Deleting DC Operating Point Parameters**

To delete DC operating point parameters, do the following:

- Right-click on the row for a parameter and choose *Delete*.

To delete multiple parameters, hold down the *Shift* key (for contiguous selection) or the *Ctrl* key (for noncontiguous selection), click the rows for the parameters you want to delete, then right-click and choose *Delete*.

To delete all parameters, do the following:

1. Right-click on the row for a parameter and choose *Select All*.
2. Right-click on the row for a parameter and choose *Delete*.

## **Viewing Sensitivity Data**

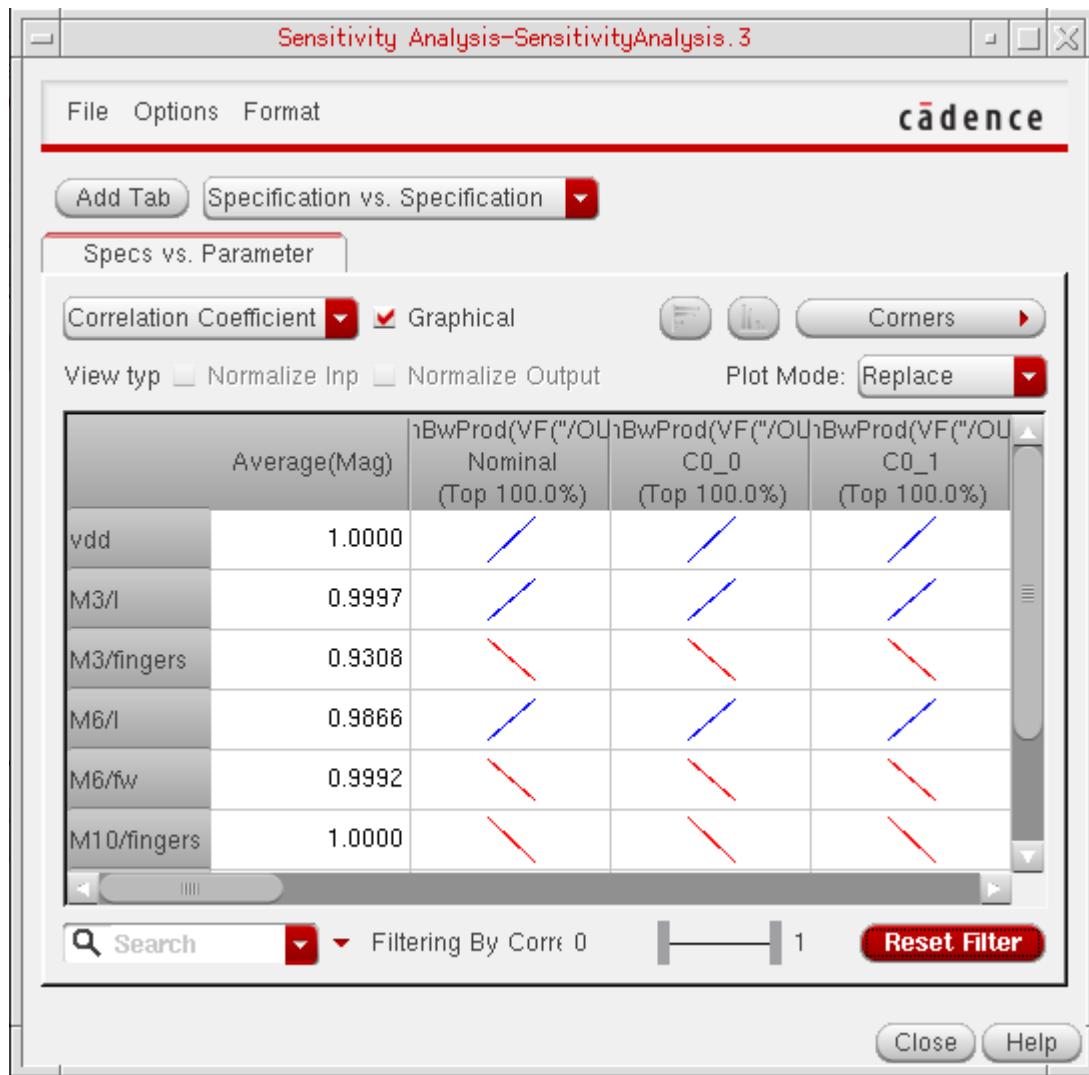
After the Sensitivity Analysis or the Monte Carlo run is complete, ADE GXL automatically opens the Sensitivity Analysis window with sensitivity analysis results displayed in it.

To view sensitivity data from a previous sensitivity analysis run, do one of the following:

- Click the *Sensitivity Results*  button on the Results tab.
- In the *History* tab on the Data View pane, right-click on the history item for a Sensitivity Analysis or Monte Carlo run and choose *Sensitivity Results*.

The Sensitivity Analysis window appears.

## **Figure 2-1 Sensitivity Analysis Window**



## Reading Data in the Sensitivity Analysis window

In the Sensitivity Analysis window, sensitivity data is displayed in a tabular format, where each data cell shows the sensitivity of a spec or parameter in the corresponding column to the spec or parameter displayed in the corresponding row. The specs or parameters displayed in the inputs and outputs depend on the selected tab.

By default, the data shows sensitivity of each specification with respect to the available device and statistical parameters. This view is governed by the *Specs vs. Parameter* tab. To know more about the other available tabs, refer to [Viewing Sensitivity Data in Different Tabs](#).

If the analysis was run on various corners, a column is added for each corner condition. For example, in the figure shown above, data for the `gainBwProd` spec is shown for all the three corners, nominal, C0\_0, and C0\_2.

**Note:** If you disable any test for a corner in the **Corners Setup** form, specification columns for that test and corner combination are not shown in the Sensitivity Analysis window.

There are various ways in which you can customize the way in which the sensitivity data is displayed. For more details on how to customize the view, refer to [Working with Sensitivity Data](#).

For details on how to plot sensitivity results, refer to [Plotting Sensitivity Analysis Results](#).

## **Viewing Sensitivity Data in Different Tabs**

The Sensitivity Analysis window provides the following tabs for viewing sensitivity data:

- Specs vs. Specs
- Specs vs. Parameter
- Specs vs. Stat Param
- Specs vs. DC Ops
- DC Ops vs. Parameter
- DC Ops vs. Stat Param
- Stat Param vs. Stat Param

In these tabs, the input factor is displayed in the rows and the output is displayed in the columns. For example, in the Specs vs. Parameter tab shown in [Figure 2-1](#) on page 55, parameters are input factors and specifications are the output. Therefore, parameters are displayed in the rows and specifications are displayed in the columns. For information about working with the data in the tabs, see [Working with Sensitivity Data](#) on page 60.

**Note:** The text `N.A.` in a cell in the Sensitivity Analysis window indicates that the sensitivity value could not be calculated because of some reason. You can place the mouse pointer on the cell to view the reason in a tooltip.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

These tabs are described below:

Tab Name	Description
Specs vs. Specs	<p>Displays the sensitivity of each specification with respect to other specifications.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>1. From the drop-down list next to the <i>Add Tab</i> button, choose <i>Specification vs. Specification</i>.</li><li>2. Click the <i>Add Tab</i> button to display the Specs vs. Specs tab.</li></ol>
Specs vs. Parameter	<p>Displays the sensitivity of each specification with respect to device and statistical parameters.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>1. From the drop-down list next to the <i>Add Tab</i> button, choose <i>Specification vs. Parameter</i>.</li><li>2. Click the <i>Add Tab</i> button to display the Specs vs. Parameter tab.</li></ol> <p><b>Note:</b> The sensitivity of statistical parameters is displayed in this tab only if sensitivity data for statistical parameters exists in the results database. See also, <a href="#">Hiding or Showing Statistical Parameters</a> on page 77.</p>
Specs vs. Stat Param	<p>Displays the sensitivity of each specification with respect to statistical parameters.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>1. From the drop-down list next to the <i>Add Tab</i> button, choose <i>Specification vs. Statistical Parameter</i>.</li><li>2. Click the <i>Add Tab</i> button to display the Specs vs. Stat Param tab.</li></ol> <p><b>Note:</b> The <i>Specification vs. Statistical Parameter</i> option is available only if sensitivity data for statistical parameters exists in the results database.</p>

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

Tab Name	Description
Specs vs. DC Ops	<p>Displays the sensitivity of each specification with respect to DC operating point parameters.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>From the drop-down list next to the <i>Add Tab</i> button, choose <i>Specification vs. DC OP Point Parameter</i>.</li></ol> <p><b>Note:</b> The <i>Specification vs. DC OP Point Parameter</i> option is available only if sensitivity data for DC operating point parameters exists in the results database.</p> <ol style="list-style-type: none"><li>Click the <i>Add Tab</i> button to display the Specs vs. DC Ops tab.</li></ol>
DC Ops vs. Parameter	<p>Displays the sensitivity of each DC operating point parameter with respect to device and statistical parameters.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>From the drop-down list next to the <i>Add Tab</i> button, choose <i>DC OP Point Parameter vs. Parameter</i>.</li></ol> <p><b>Note:</b> The <i>DC OP Point Parameter vs. Parameter</i> option is available only if sensitivity data for DC operating point parameters exists in the results database.</p> <ol style="list-style-type: none"><li>Click the <i>Add Tab</i> button to display the DC Ops vs. Parameter tab.</li></ol> <p><b>Note:</b> The sensitivity of statistical parameters is displayed in this tab only if sensitivity data for statistical parameters exists in the results database. See also, <a href="#">Hiding or Showing Statistical Parameters</a> on page 77.</p>

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

Tab Name	Description
DC Ops vs. Stat Param	<p>Displays the sensitivity of each DC operating point parameters with respect to statistical parameters.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>From the drop-down list next to the <i>Add Tab</i> button, choose <i>DC OP Point Parameter vs. Statistical Parameter</i>.</li></ol> <p><b>Note:</b> The <i>DC OP Point Parameter vs. Statistical Parameter</i> option is available only if sensitivity data for DC operating point parameters and statistical parameters exists in the results database.</p> <ol style="list-style-type: none"><li>Click the <i>Add Tab</i> button to display the DC Ops vs. Stat Param tab.</li></ol>
Stat Param vs. Stat Param	<p>Displays the sensitivity of each statistical parameter with respect to other statistical parameters.</p> <p>Do the following to display the tab:</p> <ol style="list-style-type: none"><li>From the drop-down list next to the <i>Add Tab</i> button, choose <i>Statistical Parameter vs. Statistical Parameter</i>.</li></ol> <p><b>Note:</b> The <i>Statistical Parameter vs. Statistical Parameter</i> option is available only if sensitivity data for statistical parameters exists in the results database.</p> <ol style="list-style-type: none"><li>Click the <i>Add Tab</i> button to display the Stat Param vs. Stat Param tab.</li></ol>

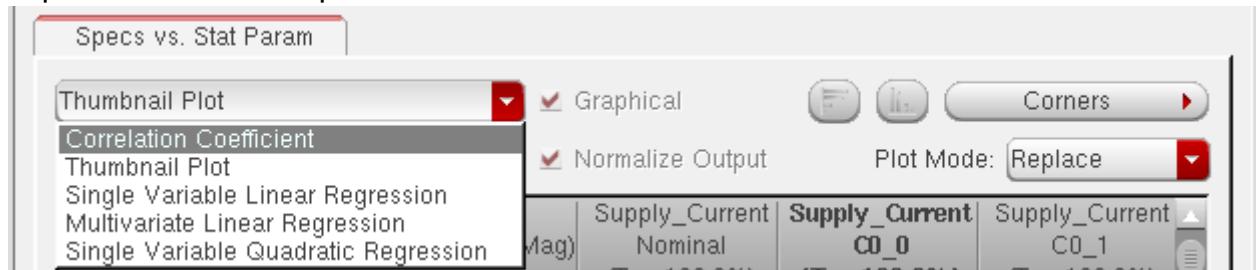
## Working with Sensitivity Data

The following topics describe how you can work with the data in the Sensitivity Analysis window:

- [Changing the Type and Format of Data To View on page 60](#)
- [Changing the Data View Type on page 71](#)
- [Filtering Data Based on Correlation Coefficient on page 72](#)
- [Filtering Data Based on Search Criteria on page 74](#)
- [Sorting Data on page 75](#)
- [Hiding Corner Data on page 76](#)
- [Hiding Specification Data on page 76](#)
- [Hiding or Showing the Detailed Results for Corners on page 76](#)
- [Hiding or Showing the Detailed Results for Specifications on page 77](#)
- [Hiding or Showing Statistical Parameters on page 77](#)
- [Showing All Hidden Data on page 77](#)
- [Highlighting Associated Devices in the Schematic on page 77](#)
- [Viewing Results in Multiple Sensitivity Analysis Windows on page 78](#)
- [Changing Number Format on page 78](#)

### Changing the Type and Format of Data To View

By default, the Sensitivity Analysis window shows correlation data in graphical format. You can also view regression and normalized regression data by selecting an option from the drop-down list at the top of a tab.



By default, the Sensitivity Analysis window shows both correlation and regression data in the graphical format. This format provides an overview of the data in a thumbnail form.

To change the format of data to the non-graphical (numeral) format:

- Clear the *Graphical* checkbox next to the drop-down list at the top of a tab.

For more details on different types and formats of data in the Sensitivity Analysis window, see:

- [Viewing Correlation Data](#) on page 61
- [Viewing Correlation Data in Graphical Format](#) on page 62
- [Viewing Thumbnail Plots](#) on page 64
- [Viewing Regression Data](#) on page 66
- [Viewing Normalized Regression Data](#) on page 67
- [Viewing Regression Data in Graphical Format](#) on page 70

### ***Viewing Correlation Data***

Correlation data tells how correlated each goal is to that variable. If the correlation is close to 1 (or -1), then they are highly correlated. However, if the correlation is 0, then the goal and variable are not correlated. The correlation value is always between -1 and 1.

To view correlation data:

- Choose **Correlation Coefficient** from the drop-down list at the top of a tab.

The correlation data appears. By default, this data is in the graphical format. For more details on this, refer to [Viewing Correlation Data in Graphical Format](#) on page 62. To view the correlation data in numeral format, clear the *Graphical* checkbox.

Correlation data in numeral format is displayed as shown in the following figure.

**Figure 2-2 Correlation Data in Sensitivity Analysis window**

	Average(Mag)	avg_vt_default (Top 100.0%)	avg_vt1_default (Top 100.0%)	avg_vt2_default (Top 100.0%)
global:PiRho	N.A.	N.A.	N.A.	N.A.
global:CAP	0.9908	0.9908	0.9908	0.9908
global:pnpbeta	0.98663	-0.98663	-0.98663	-0.98663
global:PbRho	N.A.	N.A.	N.A.	N.A.
global:npnbeta	0.99578	-0.99578	-0.99578	-0.99578
mismatch:I7.Q4:pnpbeta	0.99902	-0.99902	-0.99902	-0.99902

The first column lists the variables, and the *Average(Mag)* column displays the average of absolute values of correlation coefficients for each specification. Each cell in the other columns in the table displays the correlation values. Positive correlation values are displayed in blue color and negative values are displayed in red color.

#### ***Viewing Correlation Data in Graphical Format***

When the *Graphical* checkbox is selected, the correlation data is displayed as shown in the following figure.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

**Figure 2-3 Correlation Data Displayed in the Graphical Format**

The screenshot shows a software interface for sensitivity analysis. At the top, there's a toolbar with buttons for 'Specs vs. Stat Param' (highlighted), 'Correlation Coefficient' (selected), 'Graphical' (checked), and other options like 'Normalize Input' and 'Normalize Output'. Below the toolbar is a table with six rows and five columns. The first column lists variables: 'global:PiRho', 'global:CAP', 'global:pnpbeta', 'global:PbRho', 'global:npnbeta', and 'mismatch:I7.Q4:pnpbeta'. The second column, 'Average(Mag)', contains values: 'N.A.', '0.9908', '0.98663', 'N.A.', '0.99578', and '0.99902'. The third column, 'avg\_vt\_default (Top 100.0%)', has 'N.A.' in all cells. The fourth column, 'avg\_vt1\_default (Top 100.0%)', also has 'N.A.' in all cells. The fifth column, 'av (Top 100.0%)', has 'N.A.' in all cells. Each cell in the table contains a small line icon representing the relationship between variables. A search bar and a filtering button are at the bottom.

Specs vs. Stat Param	Average(Mag)	avg_vt_default (Top 100.0%)	avg_vt1_default (Top 100.0%)	av (Top 100.0%)
global:PiRho	N.A.	N.A.	N.A.	N.A.
global:CAP	0.9908			N.A.
global:pnpbeta	0.98663			N.A.
global:PbRho	N.A.	N.A.	N.A.	N.A.
global:npnbeta	0.99578			N.A.
mismatch:I7.Q4:pnpbeta	0.99902			N.A.

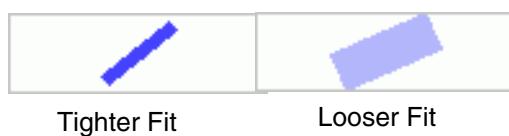
The first column lists the variables, and the *Average(Mag)* column displays the average of the fit values for each sensitivity in the row. Each cell in the other columns in the table displays a symbol that portrays:

- whether the relationship is positive or negative
- how tightly the data fits to a line

The polarity of the relationship is indicated by the direction of the slope of the line. A blue line that slopes from the lower left to upper right is positive. A red line that slopes from the upper left to the lower right is negative.



ADE GXL illustrates how tightly the data fits to a line using the width and intensity of the shape. The narrower and brighter the line, the tighter the fit.

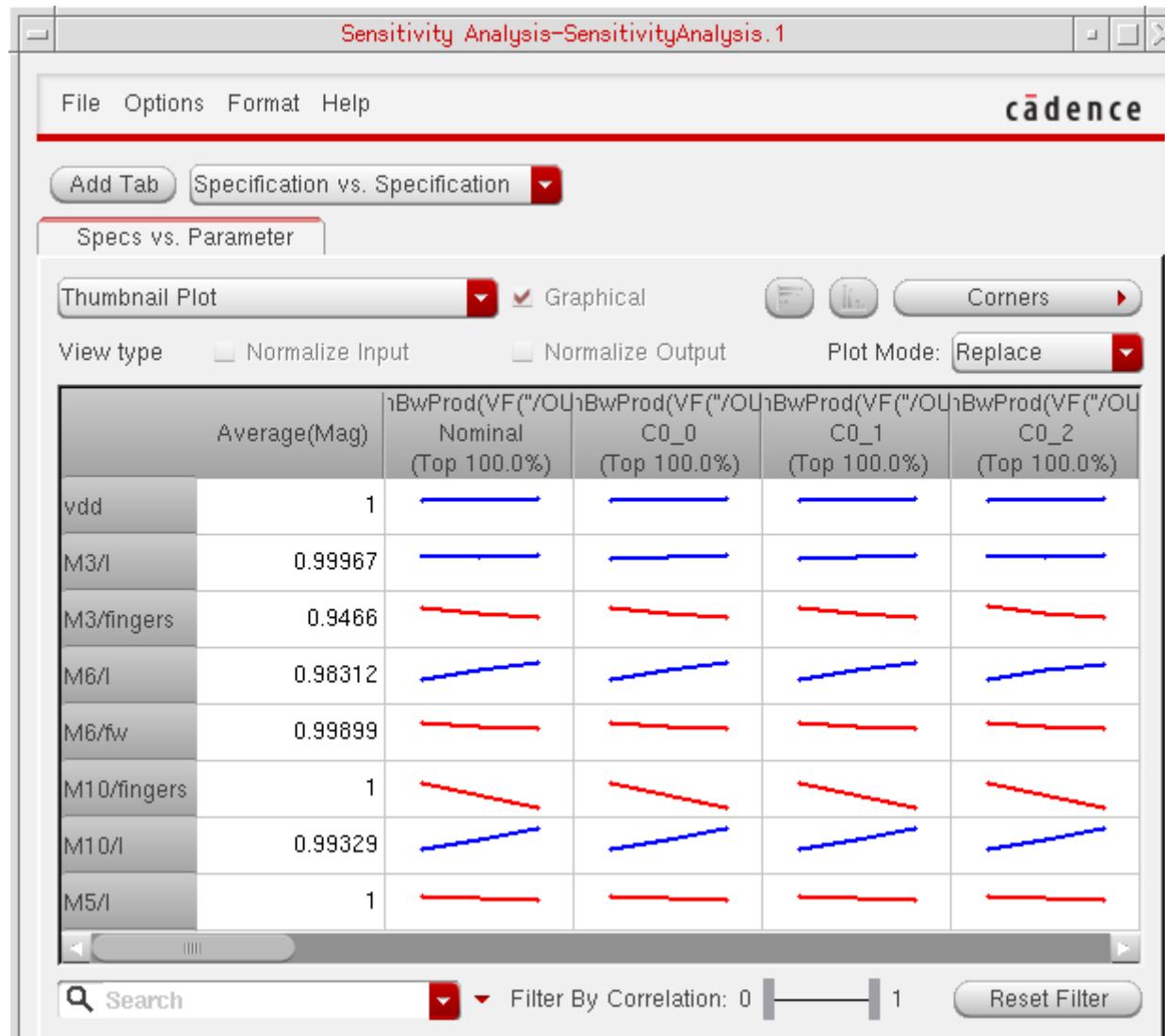


### **Viewing Thumbnail Plots**

The default graphical view is the correlation data, which shows the positive or negative relation between the inputs and outputs. However, it does not show the view of exact relationship between the two. To show a better graphical representation of the relationship between the inputs and outputs, you can choose to show thumbnail images of the actual plots of the Sensitivity Analysis or Monte Carlo run results.

To show thumbnail plots, choose **Thumbnail Plots** from the drop-down list at the top of a tab. With this option, thumbnail plots of the results data are displayed in each cell, as shown in the following figure.

**Figure 2-4 Thumbnail Plots with Line Graphs**

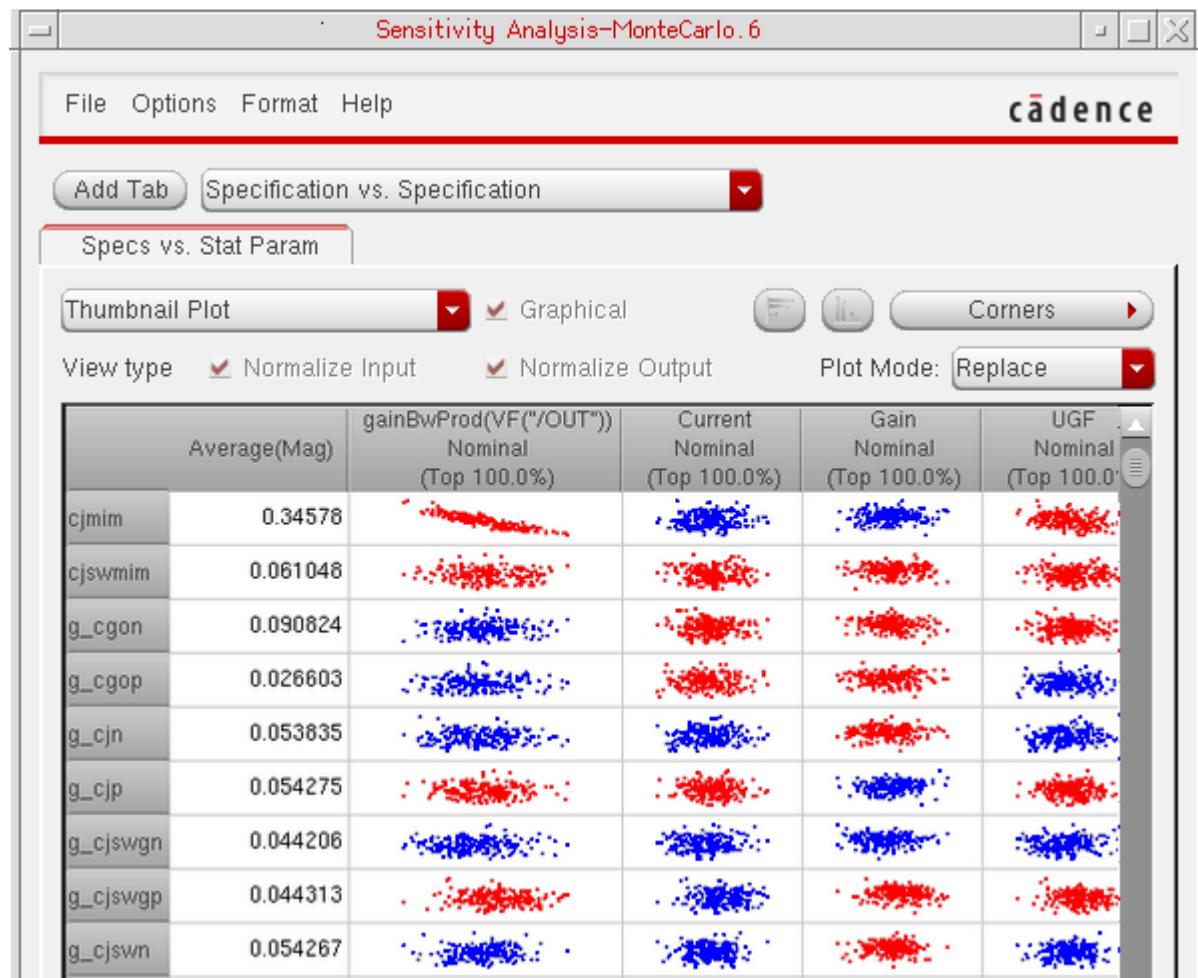


## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

By default, if the waveform contains ten points or less, a line plot is displayed, as shown in the figure above. For more than ten points, the thumbnail shows scatter plots, as shown in the following figure.

**Figure 2-5 Thumbnail Plots with Scatter Graphs**



If you want, you can specify the minimum number of points for which you want to display scatter plots by setting the `sensitivityThumbnailMaxPointsForLinePlot` environment variable.

The color of thumbnail plots depends on the sign of correlation coefficients. For positive correlation coefficients, the plots are displayed in blue and for negative values, they are displayed in red.

The thumbnail plot view gives you a view of all the plots at the same time. Click on any thumbnail to view the plot in the Virtuoso Visualization and Analysis XL window.

### ***Viewing Regression Data***

The regression data shows the best fit line or the regression line that gives an output value which is very close to the spec value.

The Sensitivity Analysis window allows you to view the following types of regression data.

- The single variable linear regression is the slope of the line between goal values and variable values that best fits the data points. Thus, a change in the variable multiplied by the regression co-efficient is the change that can be expected in the goal value.
- The single variable quadratic regression uses a quadratic equation to fit the goal values with respect to the variable values. The coefficients from this option can be considered when the correlation values are close to zero and cannot be modelled by a straight line.
- The multivariate regression gives an estimate of the linear relationship between the output and all inputs.

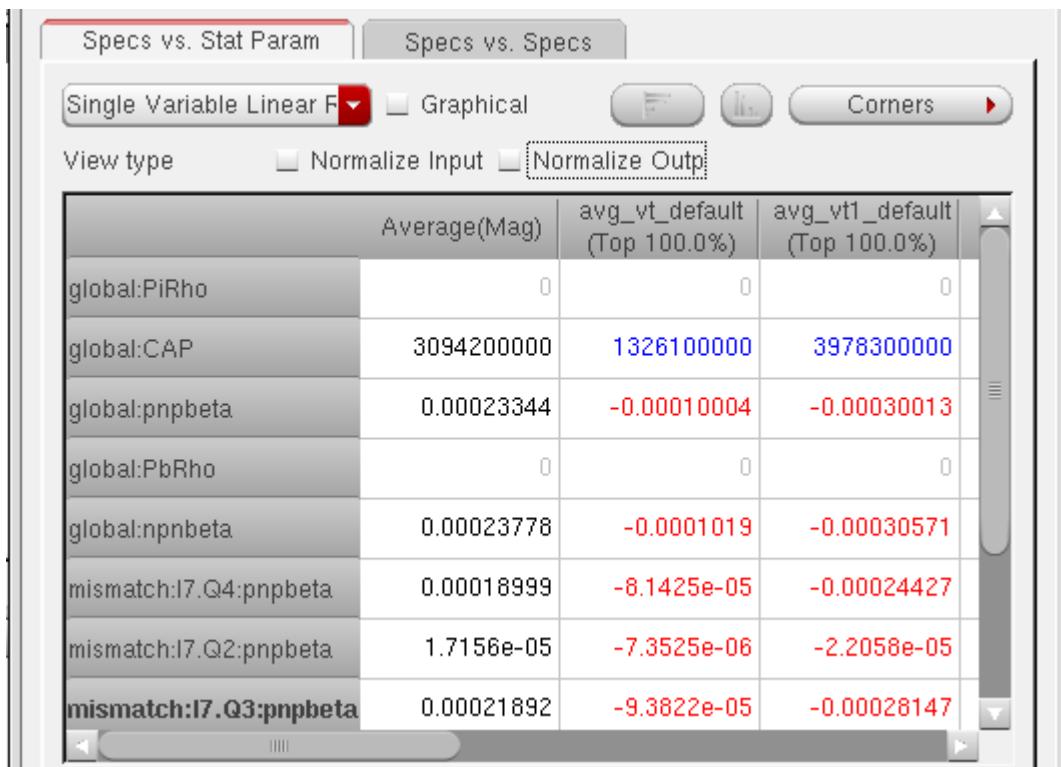
To view regression data, choose one of the following from the drop-down list at the top of a tab:

- Single Variable Linear Regression to view single variable linear regression data.
- Single Variable Quadratic Regression to view single variable quadratic regression data.
- Multivariate Linear Regression to view multivariate linear regression data.

By default, the regression data appears in the graphical format. For more details, refer to [Viewing Regression Data in Graphical Format](#) on page 70.

Clear the *Graphical* checkbox to view the data in numeral format. The regression data appears as shown in the following figure.

**Figure 2-6 Regression Data in Sensitivity Analysis window**



	Average(Mag)	avg_vt_default (Top 100.0%)	avg_vt1_default (Top 100.0%)
global:PiRho	0	0	0
global:CAP	3094200000	1326100000	3978300000
global:pnpbeta	0.00023344	-0.00010004	-0.00030013
global:PbRho	0	0	0
global:npnbeta	0.00023778	-0.0001019	-0.00030571
mismatch:I7.Q4:pnpbeta	0.00018999	-8.1425e-05	-0.00024427
mismatch:I7.Q2:pnpbeta	1.7156e-05	-7.3525e-06	-2.2058e-05
mismatch:I7.Q3:pnpbeta	0.00021892	-9.3822e-05	-0.00028147

The first column lists the variables, and the *Average(Mag)* column displays the average of absolute values of regression coefficients for each specification. Each cell in the other columns in the table displays the regression values. Positive regression values are displayed in blue color and negative values are displayed in red color.

### ***Viewing Normalized Regression Data***

Normalized regression is the value of the regression coefficient calculated when either or both the input and the output parameter values are normalized.

- When only input parameters are normalized, we say it is input normalized regression coefficient.
- When only output parameters are normalized, we say it is output normalized regression coefficient.
- When both input and output parameters are normalized, we say it is input-output normalized regression coefficient.

ADE GXL uses the following equations to normalize the data:

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

---

- To normalize input values for design variables and statistical variables:

$$X_{\text{norm}} = 2 * (X - X_{\text{min}}) / (X_{\text{max}} - X_{\text{min}})$$

- To normalize input values for Monte Carlo analysis:

$$X_{\text{norm}} = (X - X_{\text{mean}}) / X_{\text{std}}$$

- To normalize output values for design variables and statistical variables:

$$Y_{\text{norm}} = (Y - Y_{\text{mean}}) / Y_{\text{mean}}, \text{ if } Y_{\text{mean}} \neq 0$$

Normalized regression coefficients allow comparing the results between different specs for the same parameter (output normalized), same specs for different parameters (input normalized), and also between different parameter/spec combinations (input-output normalized).

To view regression data:

1. Choose *Single Variable Linear Regression*, *Single Variable Quadratic Regression*, or *Multi Variate Linear Regression* from the drop-down list at the top of a tab.

The regression data appears.

2. Do one of the following:

- To normalize the input data, select the *Normalize Input* check box.
- To normalize the output data, select the *Normalize Output* check box.
- To normalize both the input and the output data, select the *Normalize Input* and *Normalize Output* check boxes.

The regression data appears.

**Figure 2-7 Regression Data in Sensitivity Analysis window**

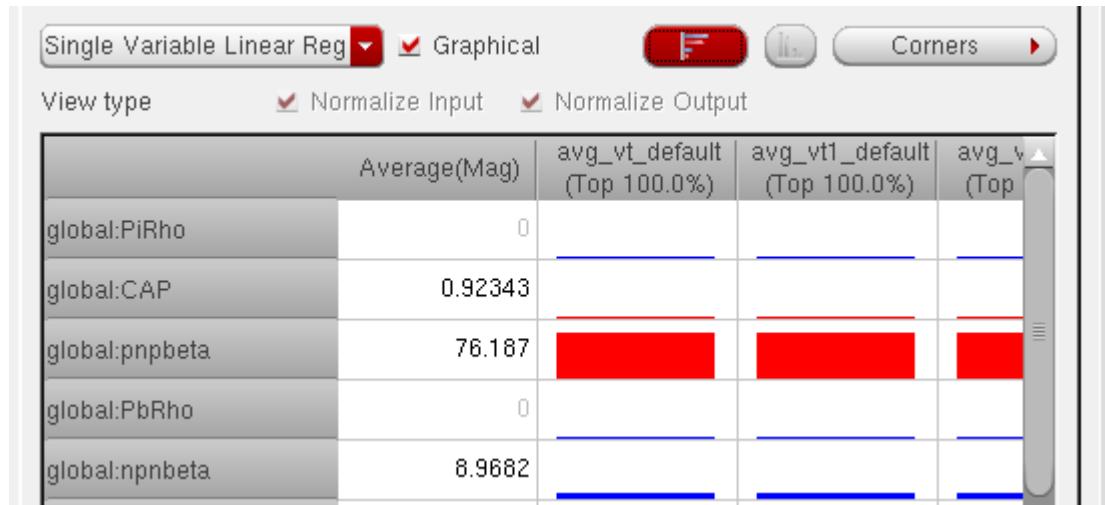
The screenshot shows the Sensitivity Analysis window with the 'Specs vs. Stat Param' tab selected. The 'Single Variable Linear Fit' button is highlighted. The table displays the following data:

	Average(Mag)	avg_vt_default (Top 100.0%)	avg_vt1_default (Top 100.0%)
global:PiRho	0	0	0
global:CAP	0.92343	-0.92343	-0.92343
global:pnpbeta	76.187	-76.187	-76.187
global:PbRho	0	0	0
global:npnbeta	8.9682	8.9682	8.9682
mismatch:I7.Q4:pnpbeta	1.1655	1.1655	1.1655
mismatch:I7.Q2:pnpbeta	0.10025	0.10025	0.10025
mismatch:I7.Q3:pnpbeta	1.3808	1.3808	1.3808

The first column lists the variables, and the *Average(Mag)* column displays the average of absolute values of normalized regression coefficients for each specification.

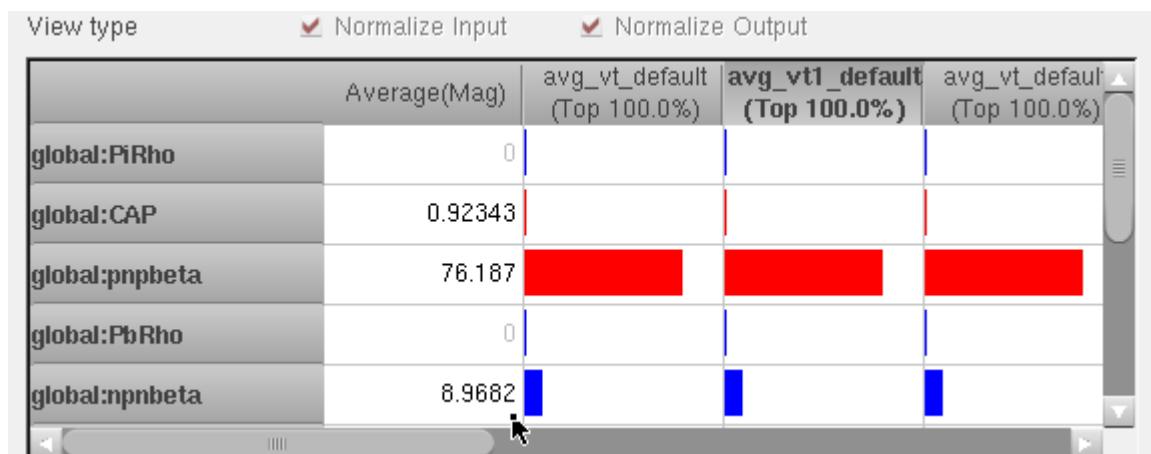
### ***Viewing Regression Data in Graphical Format***

In graphical format, the regression data is shown as bars. By default, the data is displayed as vertical bars, as shown in the following figure.



**Note:** Here, positive regression values are displayed in blue color and negative values are displayed in red color.

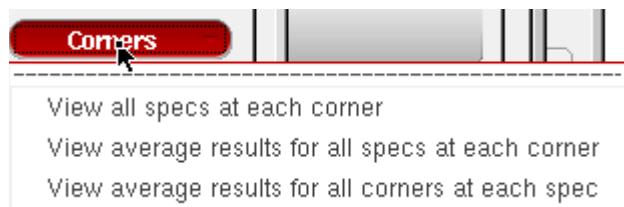
To display the regression data as horizontal bars, click the button next to the *Graphical* checkbox. The graphical data is changed as shown in the following figure.



When you view data in graphical format, the *Normalize Input* and *Normalize Output* options are automatically disabled. In addition, these two options are automatically selected when you are viewing single variable or multivariate linear regression data. For correlation data, these two options are not used.

## Changing the Data View Type

A view type specifies the data that you want to view in inputs and outputs. You can specify the data view type by using the following list at top right of the tab.



The available view types and their descriptions are listed in the following table.

<b>View Type</b>	<b>Description</b>
View all specs at each corner	View detailed results for all corners and specs. This is the default view.
View average results for all specs at each corner	View average results of all specs at each corner.
View average results for all corners at each spec	View average results at all corners for each spec.

You can plot the average results displayed by the view types listed in the table given above.

See also:

- [Hiding Corner Data](#) on page 76
- [Hiding Specification Data](#) on page 76
- [Hiding or Showing the Detailed Results for Corners](#) on page 76
- [Hiding or Showing the Detailed Results for Specifications](#) on page 77
- [Hiding or Showing Statistical Parameters](#) on page 77
- [Showing All Hidden Data](#) on page 77

## Filtering Data Based on Correlation Coefficient

With the *Filter by Correlation* slider bars, you can choose to filter out relationships that are in a range of two given correlation values. By default, all the results in the range of 0 and 1 are displayed.

You can move the slider bars to specify the correlation filter range. For example, if you move the slider bars as shown below, ADE GXL displays data with correlation coefficient values between 0.5 and 1.0.

	Average(Mag)	Supply_Current Nominal (Top 100.0%)	UGF Nominal (Top 100.0%)	Phase_Margin Nominal (Top 100.0%)
Supply_Current_Nominal	0.7348	1		0.93003
UGF_Nominal	0.53578		1	
Phase_Margin_Nominal	0.81507	0.93003		1
Open_Loop_Gain_Nominal	0.7963	-0.76623		-0.92452
area_0_Nominal				

## Filtering Data Based on Columns

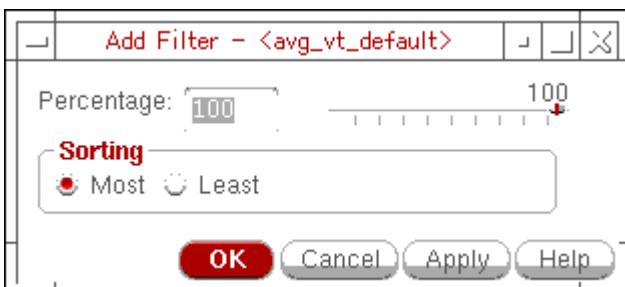
To find out the most or least sensitive parameters, you can apply a filter on a column. This will filter out parameter rows based on the value of the column data. When the correlation table is displayed, the filter is applied to the correlation data. When the regression table is displayed, the filter is applied to the regression data.

To filter data based on a column:

1. Select a column based on which you want to filter the data.
2. Right-click and choose *Add Filter*.

The Add Filter form is displayed.

**Figure 2-8 Add Filter Form**



3. Specify a percentage value.

Alternatively, use the percentage slider to specify a percentage of parameters to be filtered.

4. In the *Sorting* section, select any one of the two options, *Most* or *Least*, to display a specific percentage of top or bottom rows according to the sensitivity of parameters.

For example, if you specify 35 as the percentage value and *Most* as the sorting criteria, ADE GXL will display the top 35% of rows that show strong correlation, as shown in the following figure.

	Average(Mag)	avg_vt (Top 100.0%)	avg_vt1 (Top 35.0%)	avg_vt (Top 100.0%)
mismatch:I7.Q4:pnpbeta	0.99902	0.99902	0.99902	0.99902
mismatch:I7.Q1:npnbeta	0.99983	0.99983	0.99983	0.99983
mismatch:I7.Q0:npnbeta	0.99983	0.99983	0.99983	0.99983

**Note:** You can apply separate filters on multiple columns. In that case, ADE GXL filters data from the entire row set for each column condition separately and then displays the row intersection (rows that are common in all resulting row sets).

To remove the filters on columns:

- Right-click on a column and choose *Remove Filter*.
- Click *Reset Filter* on the Sensitivity Analysis window. This removes all filters applied on the form.

## Filtering Data Based on Search Criteria

The *Search* field allows you to filter the results in the Sensitivity Analysis window to view only the results for specific parameters. For example, if you type a string *Q* in the *Search* field, the form updates automatically to show only the results for the parameters whose names start with the string *Q*, as shown in the following figure.

A screenshot of the Virtuoso Analog Design Environment GXL User Guide. The main window shows a table of sensitivity analysis results. The columns are labeled: Average(Mag), avg\_vt (Top 100.0%), avg\_vt1 (Top 100.0%), and avg\_vt (Top 100.0%). The rows list various mismatch parameters. A search bar at the bottom left contains the letter 'Q'. To its right is a red button with a white 'X' and a dropdown menu labeled 'Filtering By Correlation' with a value of '0.5'. A 'Reset Filter' button is also present.

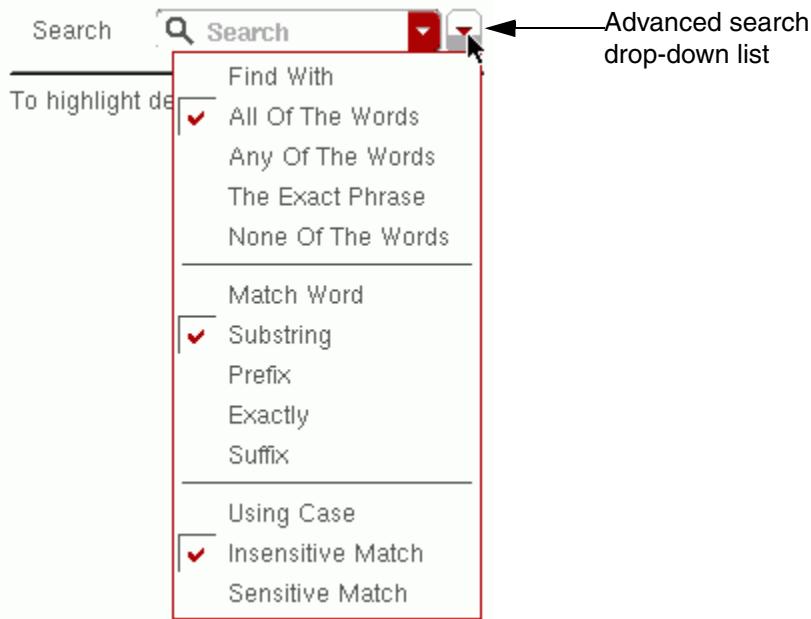
	Average(Mag)	avg_vt (Top 100.0%)	avg_vt1 (Top 100.0%)	avg_vt (Top 100.0%)
mismatch:I7.Q4:pnpbeta	0.99902	0.99902	0.99902	0.99902
mismatch:I7.Q2:pnpbeta	0.99671	0.99671	0.99671	0.99671
<b>mismatch:I7.Q3:pnpbeta</b>	0.99842	0.99842	0.99842	0.99842
mismatch:I7.Q1:pnpbeta	0.99983	0.99983	0.99983	0.99983
mismatch:I7.Q0:pnpbeta	0.99983	0.99983	0.99983	0.99983

To clear the search, click the button in the *Search* field.

Previous filters used in the current session are accessible from the drop-down list in the *Search* field.



The *Advanced* search drop-down list next to the *Search* field provides commonly used search options to refine your search results.



For more information on the *Search* field, see [The Search Toolbar](#) in the *Virtuoso Schematic Editor XL User Guide*.

## Resetting Filters

To reset all the filters applied on sensitivity results, click *Reset Filter* next to the *Filter by Correlation* slider bar.

ADE GXL removes all the applied filters and displays all the data for that view.

## Sorting Data

To sort data in a row or a column in the Sensitivity Analysis window:

- Right-click on a row or column heading and choose any of the following commands, as required:
  - *Sort by value*: This command sorts all the data based on the value in each cell.
  - *Sort by absolute value*: This command sorts all the data based on the absolute value in each cell in the selected column or row.

Data is always sorted based on normalized sensitivity regression coefficients.

## Hiding Corner Data

You can do the following to hide corner data:

- To hide the data for a specific corner, right-click on the column for the corner and choose *Hide Corners*.
- To hide the data for all corners other than a specific corner, right-click on the column for the corner you don't want to hide and choose *Hide Other Corners*.

To show all hidden data, right-click and choose *Show All*.

## Hiding Specification Data

You can do the following to hide specification data:

- To hide the data for a specification, right-click on the column for the specification and choose *Hide Specs*.
- To hide the data for all specifications other than a specific specification, right-click on the column for the specification you don't want to hide and choose *Hide Other Specs*.

To show all hidden data, right-click and choose *Show All*.

## Hiding or Showing the Detailed Results for Corners

When you are viewing average results for all specs at each corner, you can do the following to hide the detailed results for corners:

- To hide the detailed results for a corner, right-click on the column for the corner and choose *Collapse Selected Corners*.
- To hide the detailed results for all corners, right-click and choose *Collapse All Corners*.

You can do the following to show the detailed results for corners in the Corner View:

- To show the detailed results for a corner, right-click on the column for the corner and choose *Expand Selected Corners*.
- To show the detailed results for all corners, right-click and choose *Expand All Corners*.

## Hiding or Showing the Detailed Results for Specifications

When you are viewing average results for all corners at each spec, you can do the following to hide the detailed results for specifications:

- To hide the detailed results for a specification, right-click on the column for the specification and choose *Collapse Selected Specs*.
- To hide the detailed results for all specifications, right-click and choose *Collapse All Specs*.

You can do the following to show the detailed results for specifications in the Spec view:

- To show the detailed results for a specification, right-click on the column for the specification and choose *Expand Selected Specs*.
- To show the detailed results for all specifications, right-click and choose *Expand All Specs*.

## Hiding or Showing Statistical Parameters

Statistical parameters are displayed in the Specs vs. Parameter tab and the DC Ops vs. Parameter tab if the *Enable variation of Statistical Parameters* check box is selected in the Sensitivity form.

To hide the statistical parameters, right-click and choose *Hide Statistical Parameters*.

To show all hidden data, right-click and choose *Show All*.

## Showing All Hidden Data

To show all hidden data, right-click and choose *Show All*.

## Highlighting Associated Devices in the Schematic

When you right-click on parameters in the Sensitivity Analysis window and choose *Highlight on Schematic*, the associated device(s) in the schematic are highlighted.

**Note:** For Monte Carlo runs, you can highlight only the devices associated with mismatch parameters.

## **Viewing Results in Multiple Sensitivity Analysis Windows**

If you want to view the results of Sensitivity Analysis in different view modes and with different filtering conditions at the same time, you can open the same results in multiple windows. Next, you can apply different view settings and filters in different windows and compare the results.

To open the results in multiple windows, perform the following steps:

1. In the *Data View*, right-click on a history run and choose *Sensitivity Results*.

A new Sensitivity Analysis window is opened with the results of the selected history displayed in it.

2. If required, customize the view of results as per your requirements.
3. In the Sensitivity Analysis window, choose *File — New Window*.

A new Sensitivity Analysis window is opened and the same results are displayed in it. The data view settings in the new window are same as those in the first window. However, you can customize these, as required. Now, the same results are visible in two different windows with different view settings.

## **Changing Number Format**

By default, in numeral format, the number of significant figures used to display data is not fixed. In the results shown below, the data appears in different significant digits.

View type		<input type="checkbox"/> Normalize Input	<input type="checkbox"/> Normalize Output	Plot Mode: Replace		
		BwProd(VF("OU")	BwProd(VF("OU")	BwProd(VF("OU")	BwProd(VF("OU")	BwProd(VF("OU")
		Average(Mag)	Nominal (Top 100.0%)	C0_0 (Top 100.0%)	C0_1 (Top 100.0%)	C0_2 (Top 100.0%)
vdd	1	1	1	1	1	1
M3/I	0.99966	0.9997	0.99979	0.99975	0.99967	
M3/fingers	0.93078	-0.99795	-0.99808	-0.99831	-0.9966	
M6/I	0.98664	0.99103	0.99544	0.99354	0.98901	

If required, you can set a fixed number of significant digits in which you want to display the numeral data. For this, you can use the **Number Formatting Options** form.

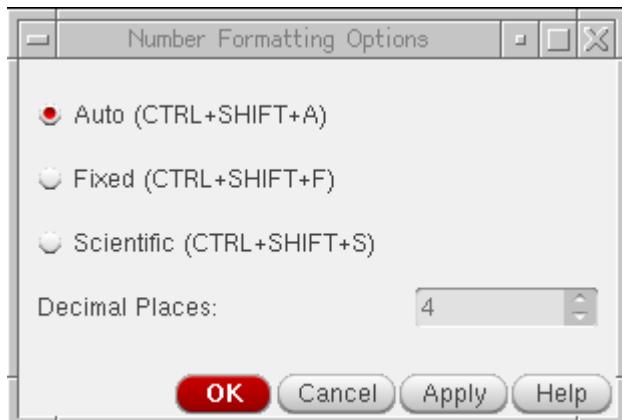
To change the number format:

1. In the Sensitivity Analysis window, choose *Format — Numbers*.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

The **Number Formatting Options** form is displayed. Note that the default format is *Auto*.



2. Select *Fixed* to display the numbers in a fixed point format.

**Note:** Similarly, you can select *Scientific* to display the numbers in scientific format.

3. In the *Decimal Places* field, specify the number of decimal places that you want to show for each number in the results.

**Note:** Decimal places can be set only upto a maximum of six digits.

4. Click *Apply* to apply the changed format.

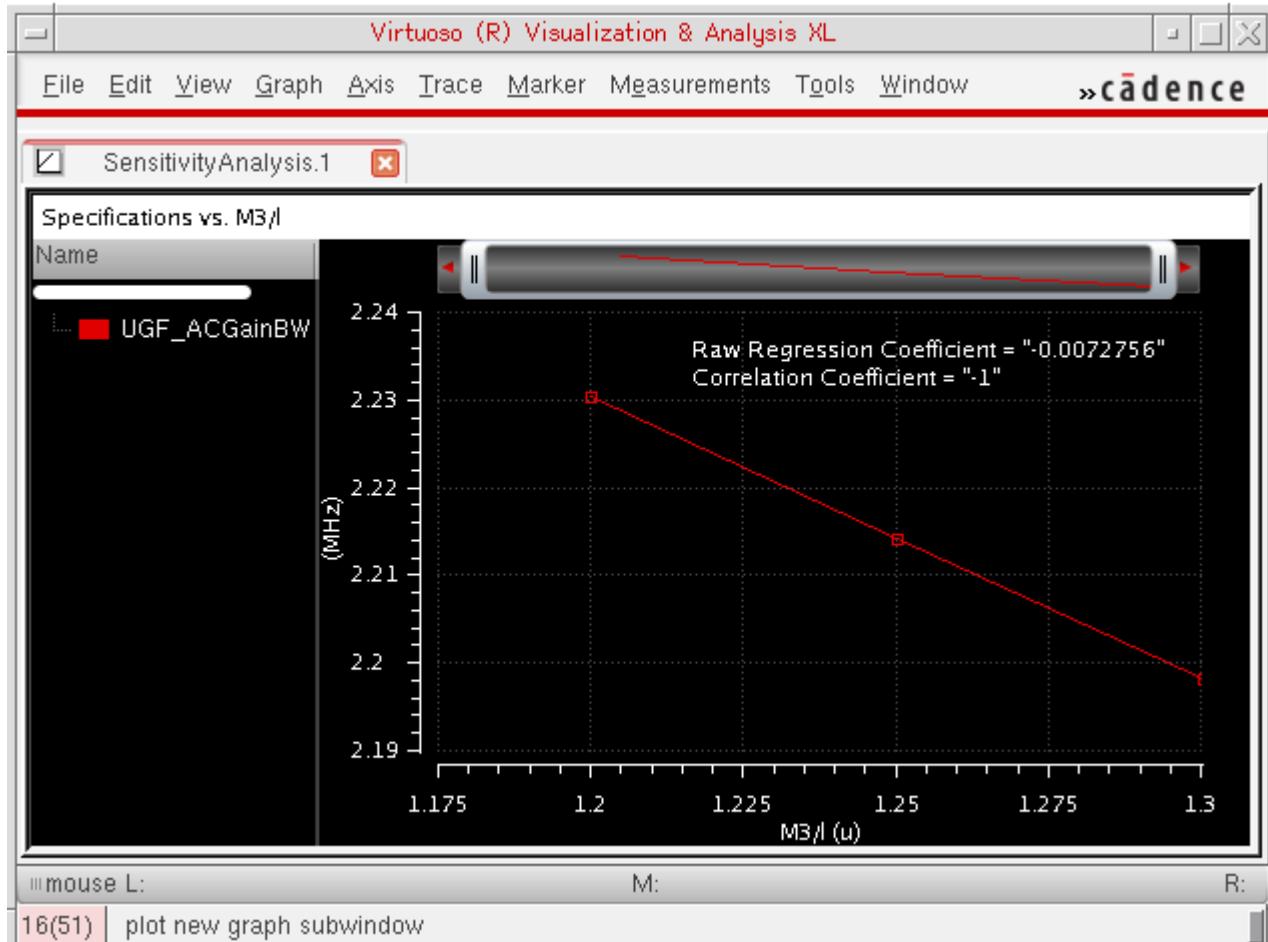
In the results shown below, the data appears in the fixed format with four decimal places.

	Average(Mag)	BwProd(VF"/OU	BwProd(VF"/OU	BwProd(VF"/OU	BwProd(VF"/OU
	Nominal (Top 100.0%)	C0_0 (Top 100.0%)	C0_1 (Top 100.0%)	C0_2 (Top 100.0%)	
vdd	1.0000	1.0000	1.0000	1.0000	1.0000
M3/I	0.9997	0.9997	0.9998	0.9998	0.9997
M3/fingers	0.9308	-0.9980	-0.9981	-0.9983	-0.9966
M6/I	0.9866	0.9910	0.9954	0.9935	0.9890
M6/fw	0.9992	-0.9996	-0.9995	-0.9996	-0.9996

## Plotting Sensitivity Analysis Results

The sensitivity analysis results can be plotted in the Virtuoso Visualization and Analysis XL window. The data can be plotted in three ways:

- Double-click any data cell in the Sensitivity Analysis results window. The correlation and regression data in the plotted graph exhibits the sensitivity of the selected specification to the corresponding parameter, as shown below.

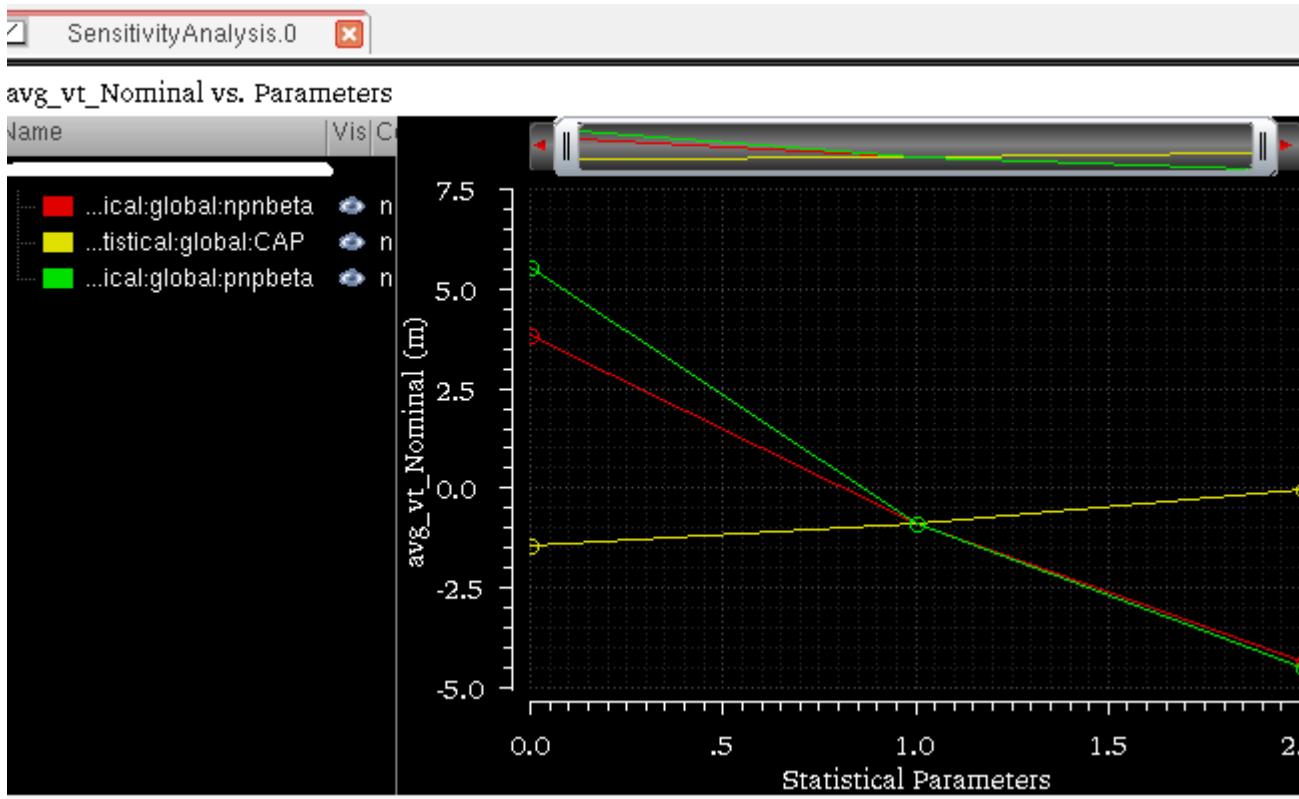


- Select two or more parameter cells for a single spec column, right-click, and choose the *Plot Across Selected Parameters* command. This displays the sensitivity of multiple

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

selected parameters to the specification represented by the corresponding column, as shown in the following figure:

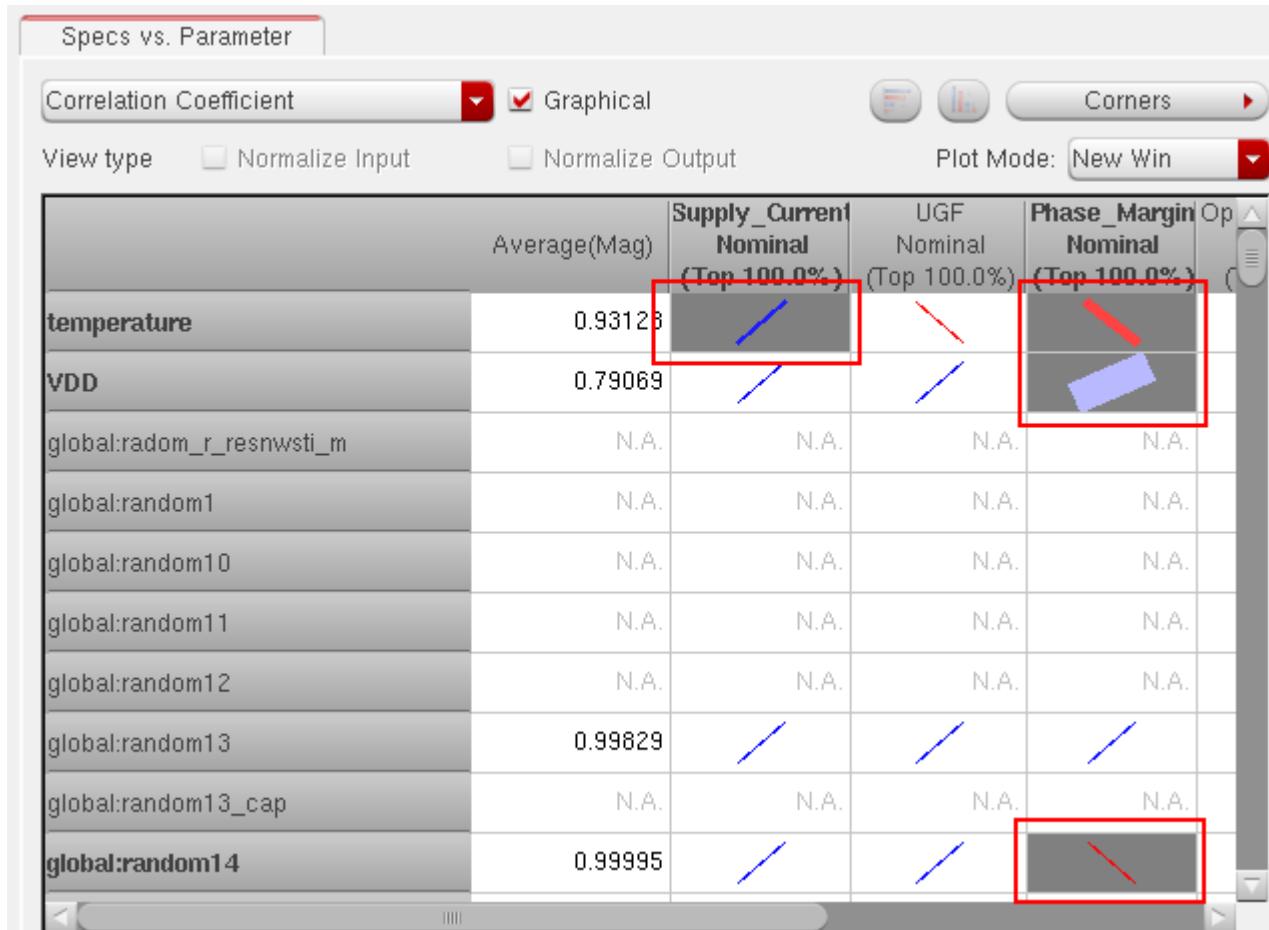


- Select two or more data cells in different spec columns, right-click, and choose the *Plot Across Parameters* command to display the sensitivity of the selected parameters to the selected specifications.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

For example, select cells for different specs and parameters, as shown in the following figure:

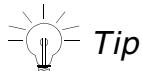
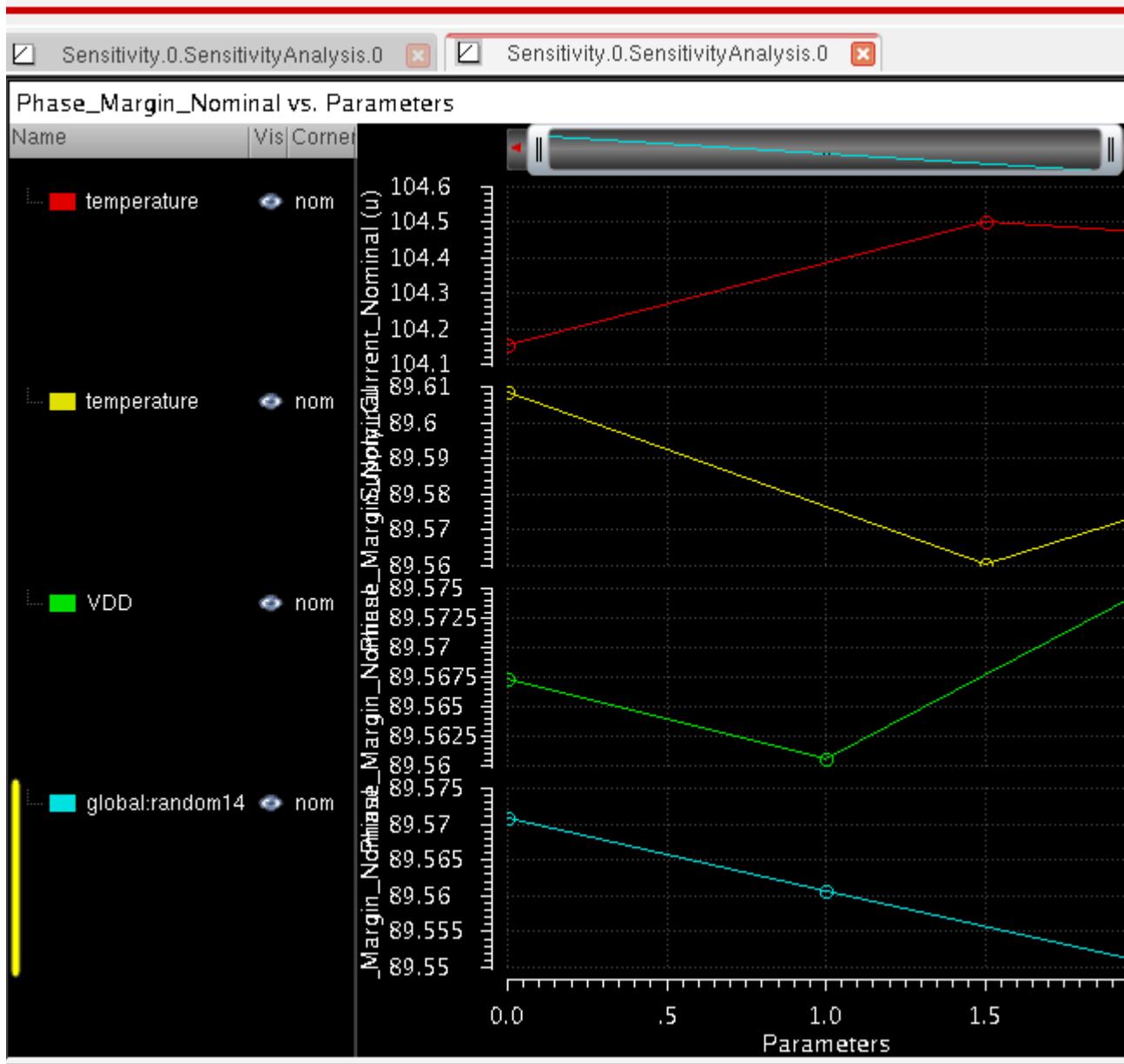


Now, right-click and choose *Plot Across Parameters*.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

In this case, the result for each parameter and spec combination is plotted in a separate strip, as shown in the figure below.



A graph is by default plotted as a continuous line. However, you can choose to plot a scatter plot by setting the `sensitivityPlotContinuousLine` environment variable in the `.cdsenv` file or in CIW as shown below.

```
envSetVal("adexl.plotting" "sensitivityPlotContinuousLine" 'boolean nil)
```

If the Virtuoso Visualization and Analysis XL window is already open for the selected history, by default, a new plot always replaces the previous plot. However, you can specify different plotting modes depending on your data analysis requirements. For more details, refer to the following sections:

- [Setting the Plotting Mode](#)
- [Displaying the ADE XL and Sensitivity Analysis Results in the Same Graph](#)
- [Plotting Average Sensitivity Analysis Results](#)
- [Displaying Spec Markers in Sensitivity Analysis Plots](#)

## Setting the Plotting Mode

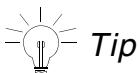
You can set the plotting mode for the sensitivity analysis results graph by using the Plotting Options form.



In the Sensitivity Analysis Results window, choose *Options – Plotting*. The Plotting Options form is displayed. In this form, the *Plot Mode* field specifies the plotting mode of the graph. By default, the default plotting mode is set to Replace.

You can select any one of the following plotting modes:

- [Replace](#)
- [Append](#)
- [New SubWin](#)
- [New Win](#)



### *Tip*

You can also set the plot mode by using the *Plot Mode* drop-down list on the Sensitivity Analysis results window.

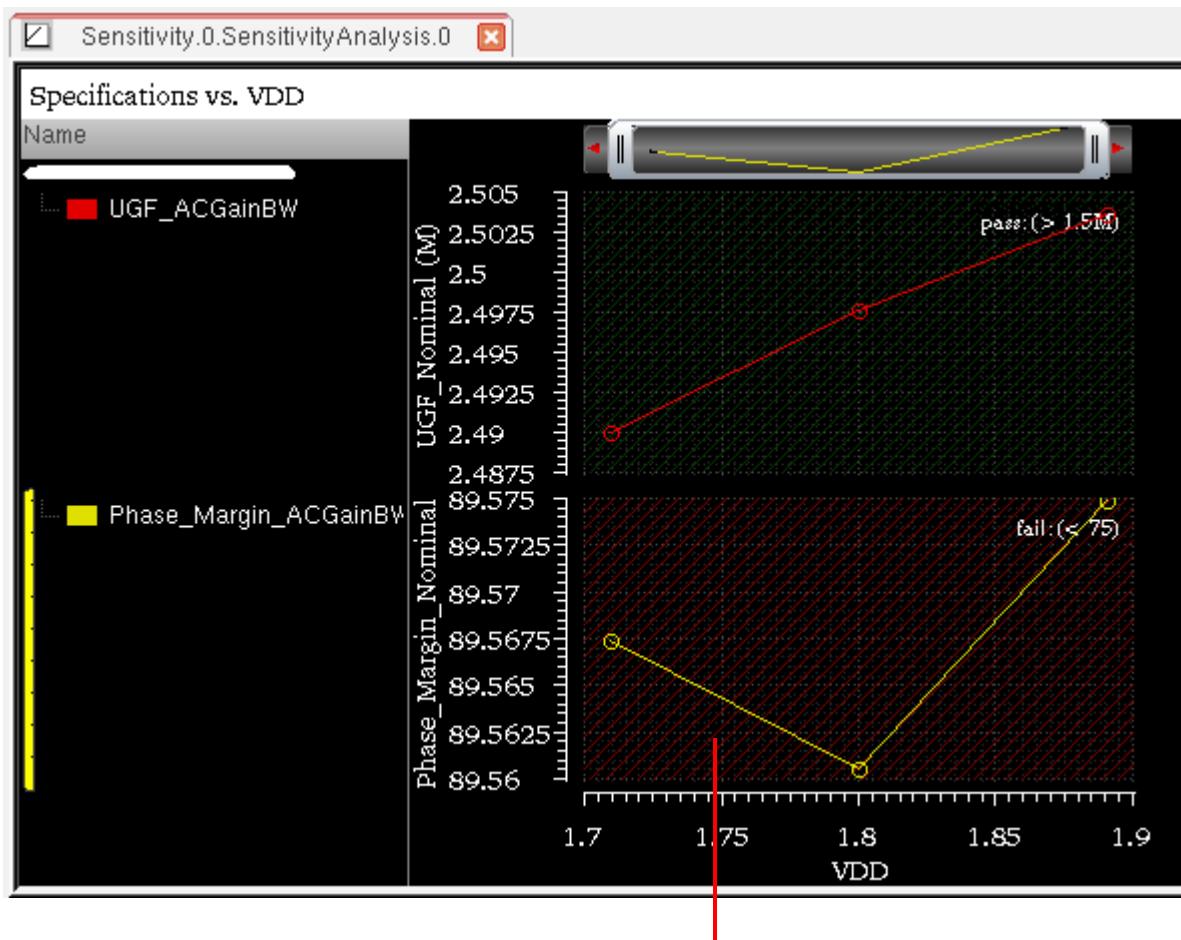
## Replace Plot Mode

In this mode, the new plot always replaces an existing graph, if any. Otherwise, a new window appears that displays the graph.

## Append Plot Mode

In this mode, if a graph is already open for the selected history, the new plot is appended to the existing graph. Otherwise, the new plot is displayed in a new window.

Consider an example. A graph is already open for the selected history and displays the sensitivity of `Open_Loop_Gain` to the parameter `M12/I`. Next, if you set the plot mode to *Append* and plot the sensitivity of `Offset_Voltage` to the parameter `M12/I`, the new plot is appended to the existing graph, as shown below.



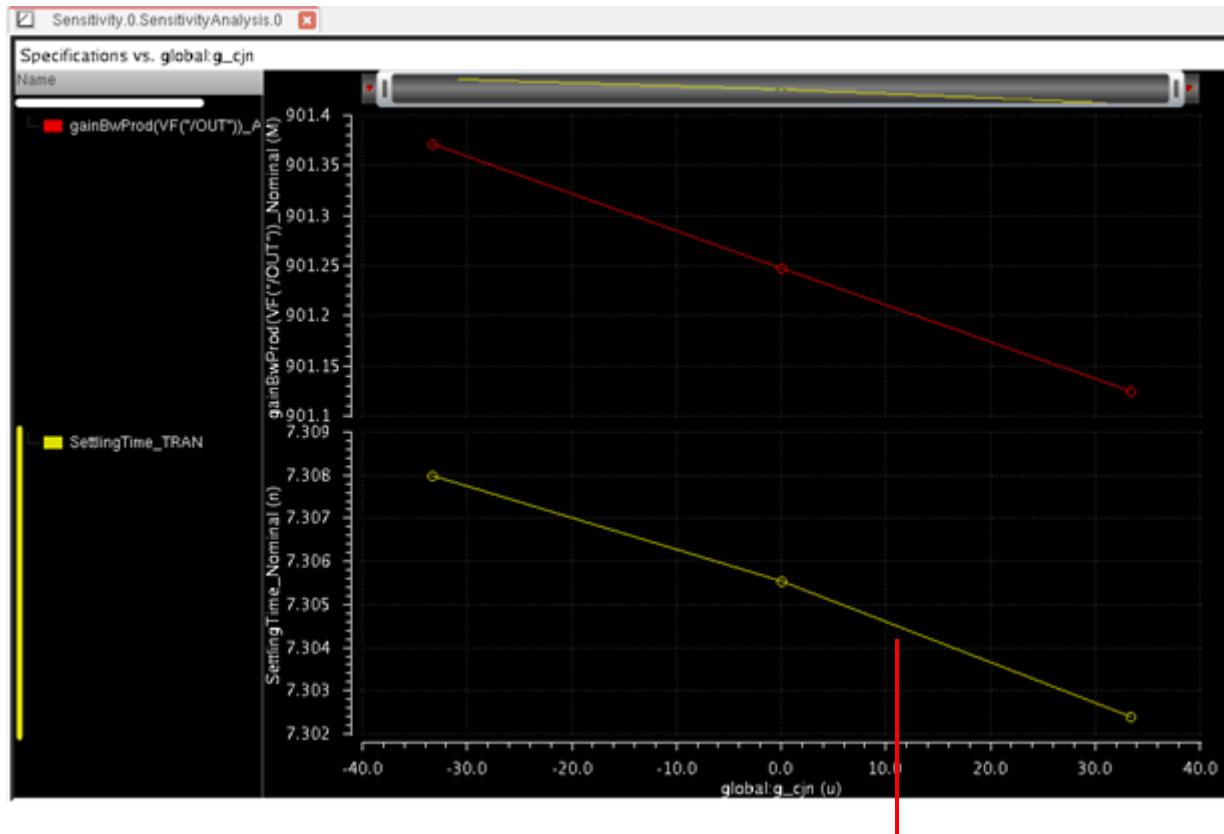
New plot appended to the existing  
graph, in a new strip

**Note:** For the plot to append to the existing graph, it is important that you plot another graph for the same parameter. ADE GXL does not retain plots with different x axes on the same graph. In the example shown above, if you plot the results for a different parameter, the graph will be plotted in a new window. However, if you select multiple results for different parameters and choose *Plot Across Parameters*, all the plots are appended to the same graph and the x axis will show Parameters.

### **Exceptions to the Append Plot Mode**

When the Plot Mode is set to Append, in the following two cases, the new plot is displayed in a new strip instead of being appended to the existing graph:

- If the new plot is for the same parameter but a different specification as that of the graph plotted earlier, the new plot is displayed in a new strip instead of being appended to the existing graph, as shown below.

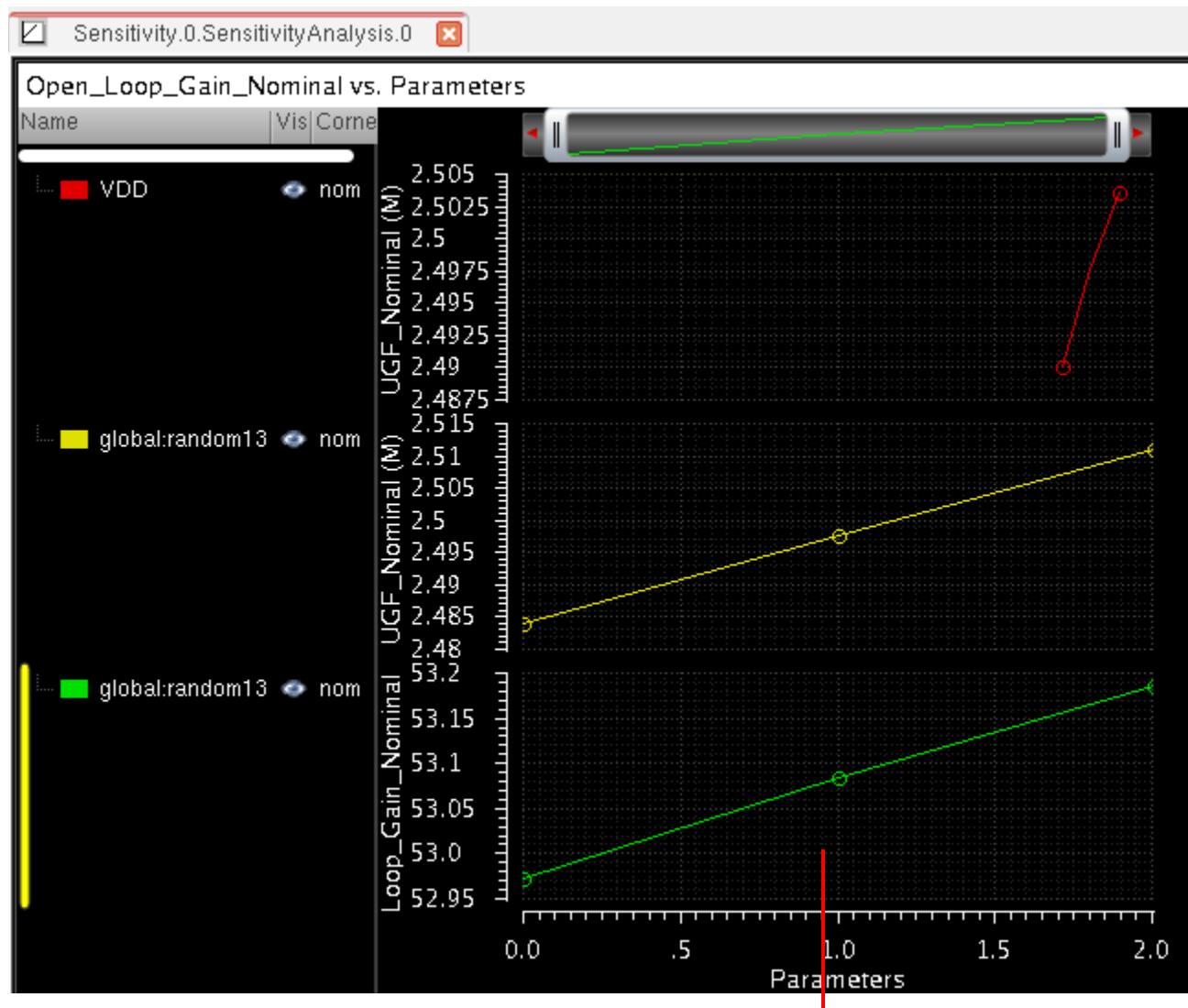


New plot appended to the  
existing graph in a new strip

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

- If you plot multiple parameters for the same specifications, and next, plot the results for new parameters, all the plots are moved to separate strips. For example, first, you plot UGF vs. VDD and UGF vs. random13. The plots are appended to the same graph because the specification UGF is common. Next, if you plot the results for another specification Loop\_Gain vs. parameter random13 by using the *Plot Across Parameters* command, all the plots will move to separate strips in the existing graph, as shown below.



**Note:** When plotting is done across different parameters, the data is normalized by using the following equations:

- For design variables and statistical variables:

$$X_{norm} = 2 * (X - X_{min}) / (X_{max} - X_{min})$$

- For Monte Carlo analysis:

$$X_{norm} = (X - X_{mean}) / X_{std}$$

### New SubWin Plot Mode

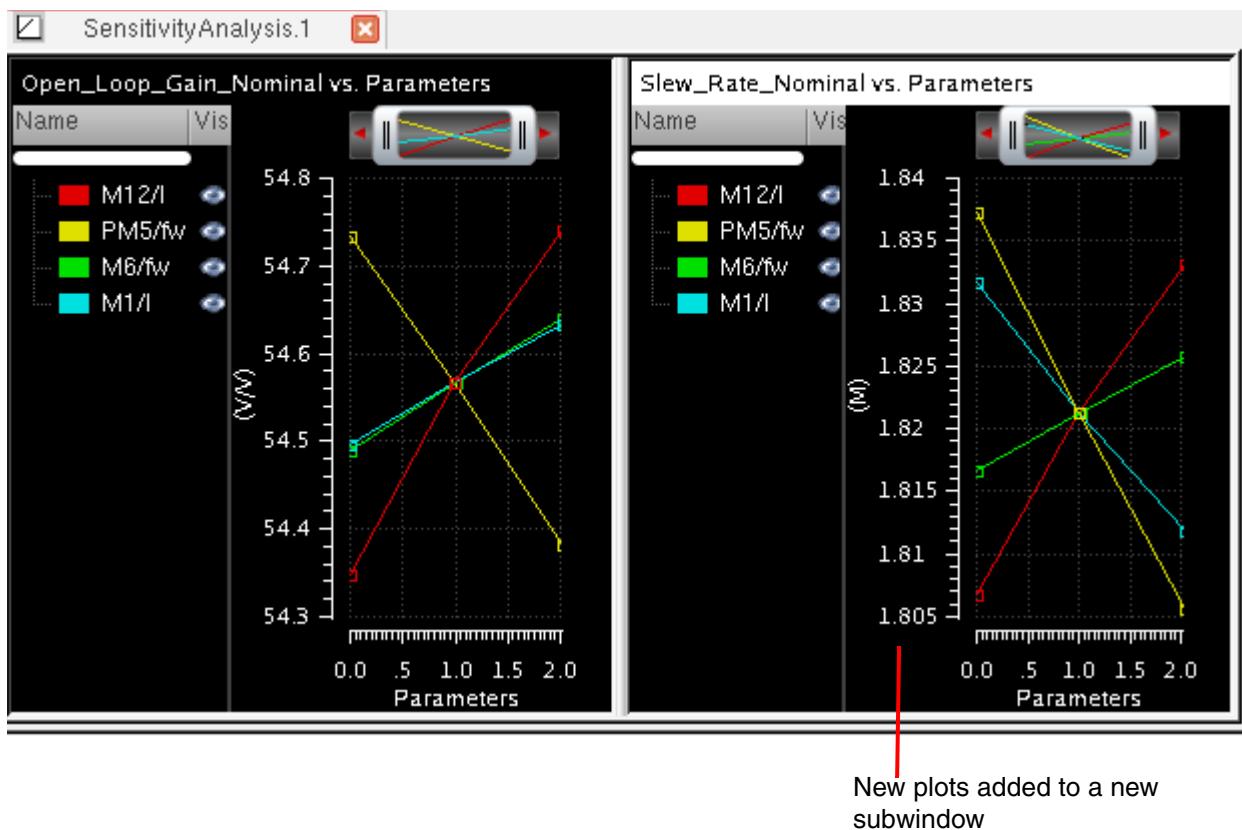
In this plot mode, if a graph is already open for the selected history, the new plot is added to a new subwindow in the existing graph.

Consider an example. You have plotted the sensitivity data for the Open\_Loop\_Gain specification to parameters M12/I, PM5/fw, M6/fw, and M1/I.

Next, you want to plot and compare the sensitivity data for Slew\_Rate to the same set of parameters. If you set the plot mode to New SubWin and plot data for these parameters, the graph appears as shown below.

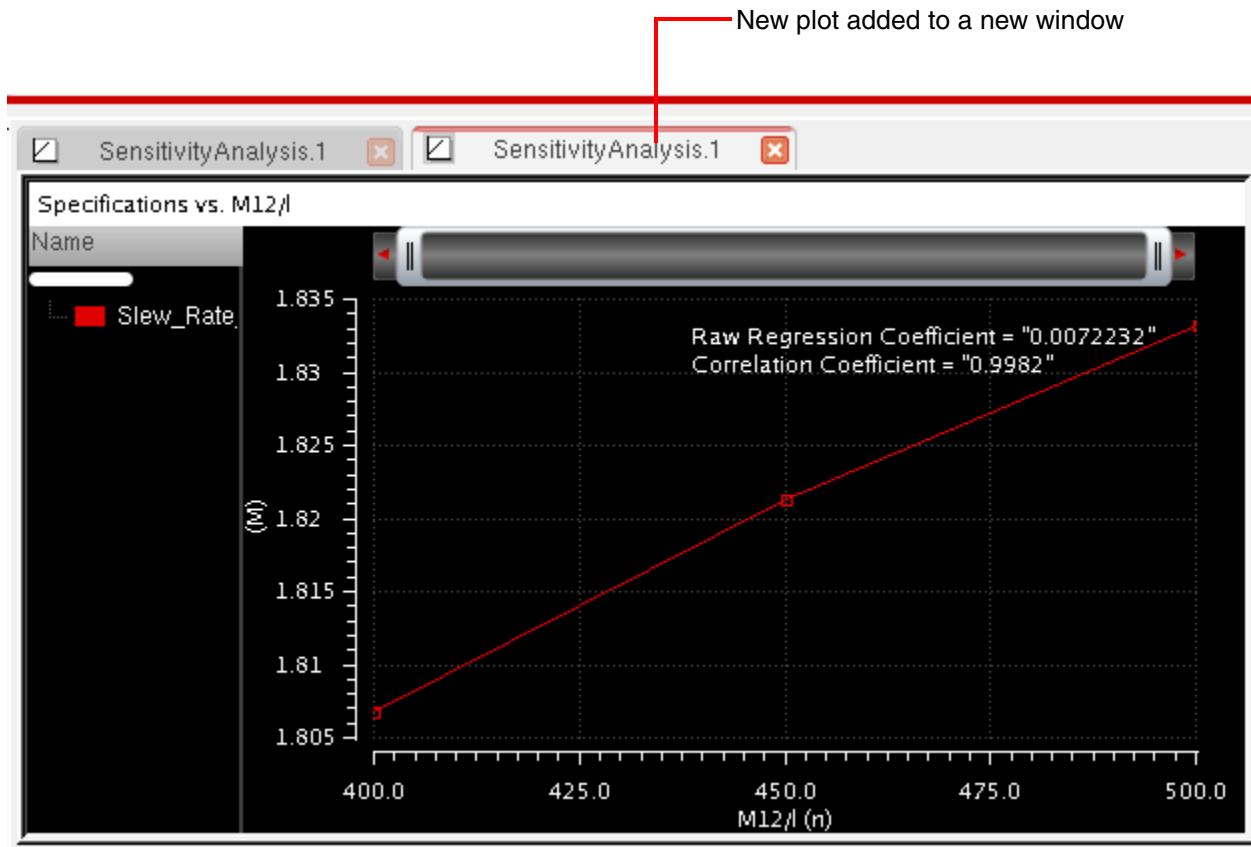
# Virtuoso Analog Design Environment GXL User Guide

## Sensitivity Analysis



### New Win Plot Mode

In this plot mode, for every new plot, a window is added to the existing Virtuoso Visualization and Analysis XL window, as shown below.



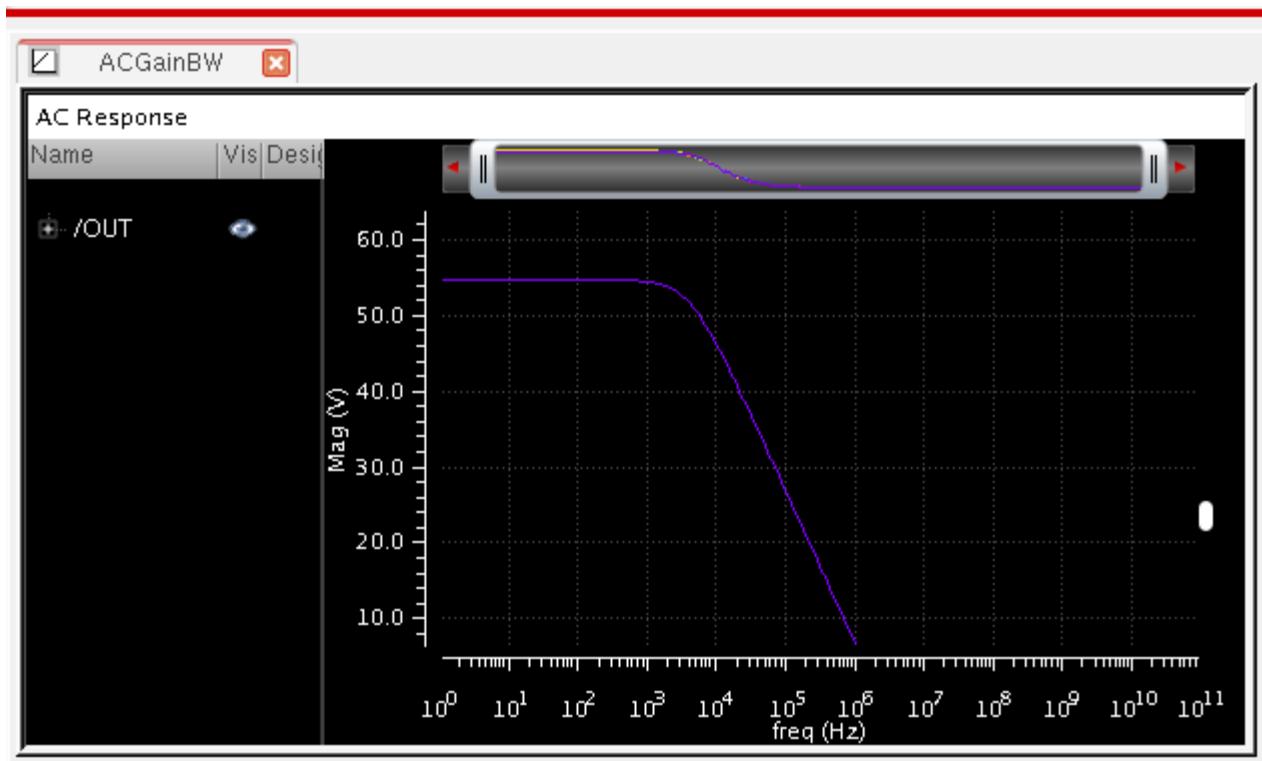
## Displaying the ADE XL and Sensitivity Analysis Results in the Same Graph

If you have already plotted results from the ADE XL window, you can choose to plot the sensitivity analysis results in the same graph. To do this, select the *Allow interaction with existing ADE XL plot windows* check box in the Plotting Options form.

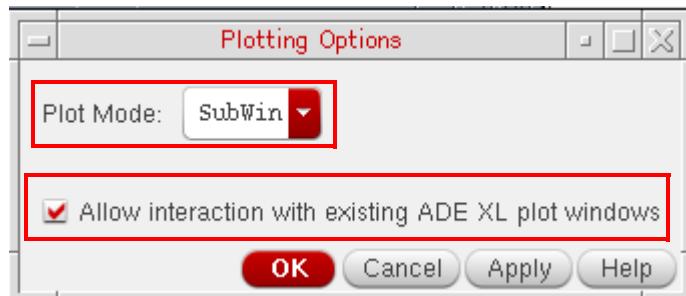
For example, when you plot an output, `ACGainBW`, from the ADE XL results tab, the graph appears as shown below.

## Virtuoso Analog Design Environment GXL User Guide

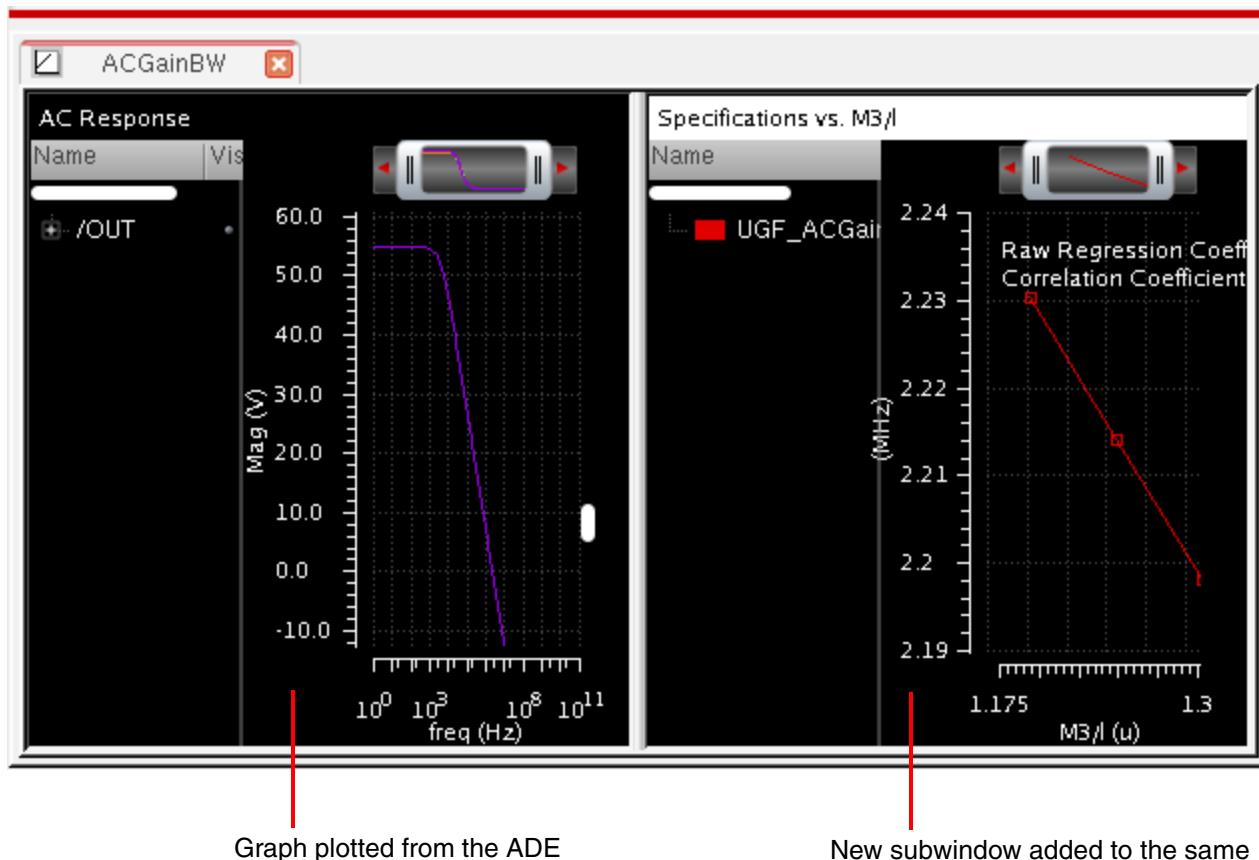
### Sensitivity Analysis



Now, if you want to plot the sensitivity results in a new subwindow of this graph to show sensitivity of UGF\_ACGainBW to the variable M3 / I, set the plotting options as shown below:



Now, if you plot the sensitivity results, a new subwindow is added as shown below.

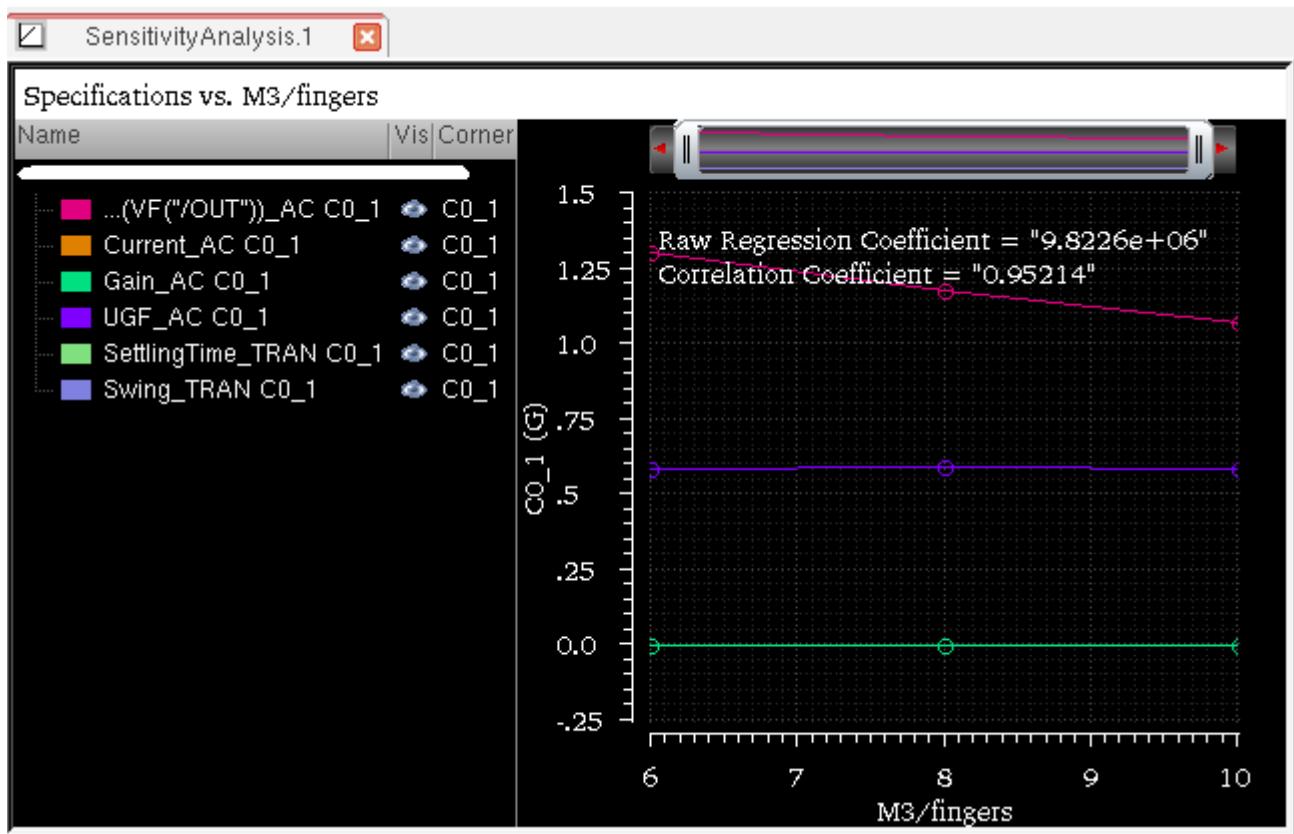


**Note:** The *Allow interaction with existing ADE XL plot windows* check box is not considered when the plot mode is set as *Append*. If this check box is selected and the *Plot Mode* is set to *Append*, the plots for the sensitivity data are not appended to the existing graph. Instead, the sensitivity data is plotted in a new window.

## Plotting Average Sensitivity Analysis Results

If you change the default Sensitivity Analysis results view and display average results of all the corners for each spec or results of all specs for each corner, you can double-click in a cell to show all the plots.

For example, if you display average results of all corners at each spec, when you double-click in a cell, the plots are displayed as shown below.



Similarly, you can plot average results of all corners at each spec.

## Displaying Spec Markers in Sensitivity Analysis Plots

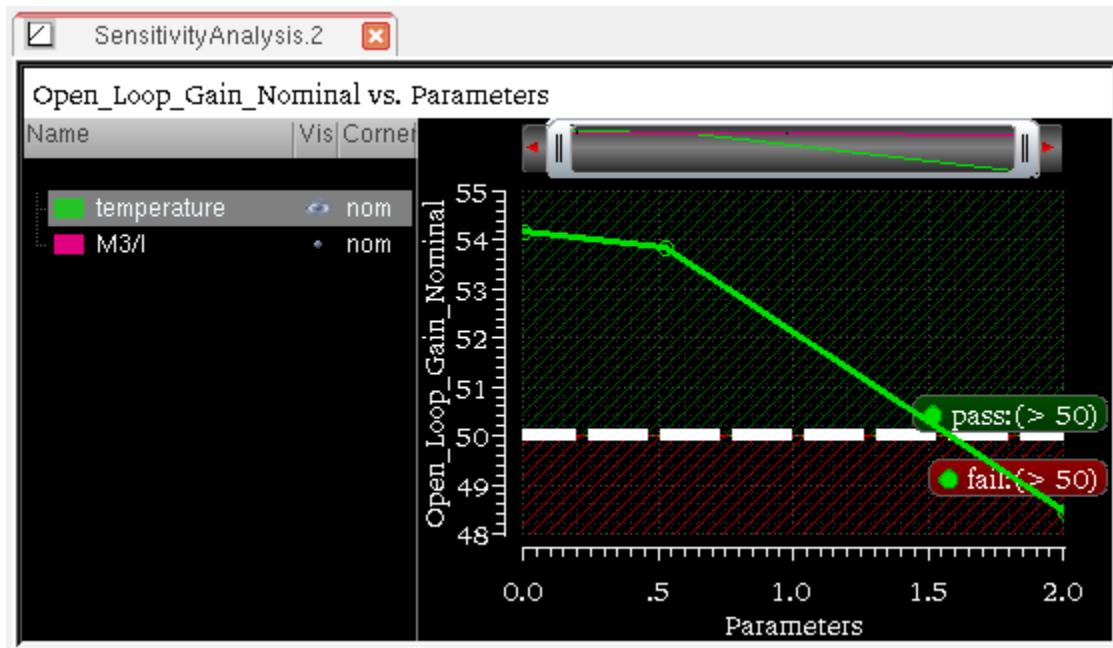
Similar to the ADE XL graphs, you can display spec markers in the graphs plotted for Sensitivity Analysis results. For this, on the Results tab of ADE XL, choose *Options — Plotting/Printing*. On the **ADE XL Plotting/Printing Options** form, select *Spec Markers*. When this option is selected, the graphs displayed from the Results tab of the Output pane show spec markers, as shown below.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis



The specs vs. parameter graphs displayed from the Sensitivity Analysis window also show spec markers, as shown below.



## Saving the Sensitivity Data

You can save the sensitivity data to a text file:

1. Choose *File – Export Results to CSV*, then do one of the following:
  - Choose *Current Tab* to save only the data in the current tab.
  - Choose *All Tabs* to save the data in all the tabs.
2. In the Export Results form, specify a path and file name.
3. Click *OK*.

## Creating Worst Case Corners

You can create and view worst case corners by using the *Create Worst Case Corners* simulation run mode. In this mode, an algorithm that uses sensitivity analysis to identify the worst case corners is run. Worst case corners are created after the sensitivity analysis is complete. Worst case corners are created for each of your design specifications based on the sensitivity result instead of enumerating all corner variable combinations.

For more information, see the following topics:

- [Specifying the Setup](#) on page 96
- [Creating Worst Case Corners](#) on page 103
- [Viewing Worst Case Corners](#) on page 103
- [Creating Worst Case Corners Interactively](#) on page 105

### Specifying the Setup

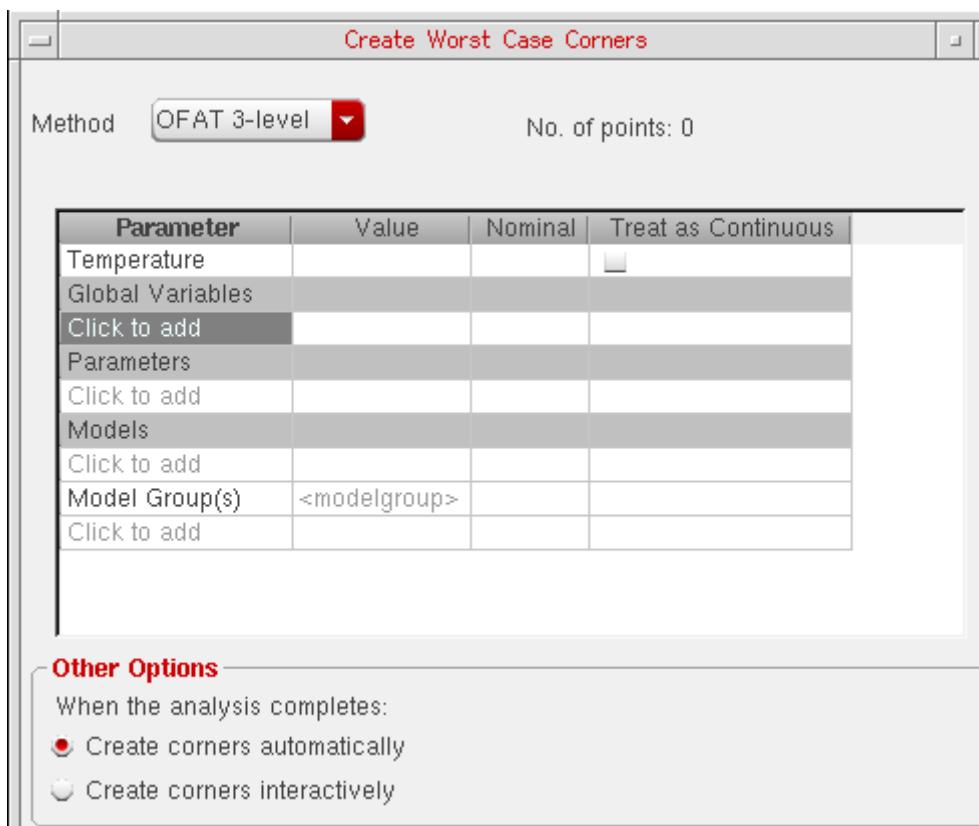
Before you create worst case corners, perform the following steps:

1. On the *Run* toolbar, in the *Select a Run Mode* drop-down list, select *Create Worst Case Corners*.
2. If you want to vary variables or parameters, ensure that you have specified variables and parameters in the Variables and Parameters assistant.
3. On the *Outputs Setup* tab, select the required spec and ensure that for each spec, you specify an objective type and a target value.

**Note:** If you specify `tolerance (tol)` or `range` objective types for the spec you choose, two worst case corners will be created. For all other objective types, one worst case corner is created.

4. Click the *Simulation Options* button.

The Create Worst Case Corners form appears.



In the Create Worst Case Corners form, specify the settings that you want to use while running the sensitivity analysis to identify the worst case corners. For more details, refer to the following section.

### Setting Up the Create Worst Case Corners Form

In the Create Worst Case Corners form, you can specify the variables and design parameters that you want to vary and the one-factor-at-a-time (OFAT) method by which you want to vary them. In addition, you can also specify how you want to create corners after the analysis is run.

**Note:** The tool varies each variable or parameter OFAT, while leaving other variables and parameters as fixed.

Perform the following steps to set up the Create Worst Case Corners form:

1. In the *Method* field, specify the method by which you want to vary the variables. The two possible methods are:

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

---

- ❑ OFAT 3-level - In this method, the tool varies each factor for three values. For example, 1.9u, 2.0u, 2.1u with nominal value equal to 2.0u. If, instead of three points, you specify a range of values, the tool chooses the following three values of the variable for simulation:
  - minimum value in the range
  - nominal value
  - maximum value in the range

For example, if the variable range is 1u:0.1u:5u and the nominal value is 3.6u, the three values used by the tool are: 1u, 3.6u, and 5u.

- ❑ OFAT Sweep - In this method, the tool varies each factor for the specified sweep values. You can either specify sweep values as a range or as a set of space-separated values.

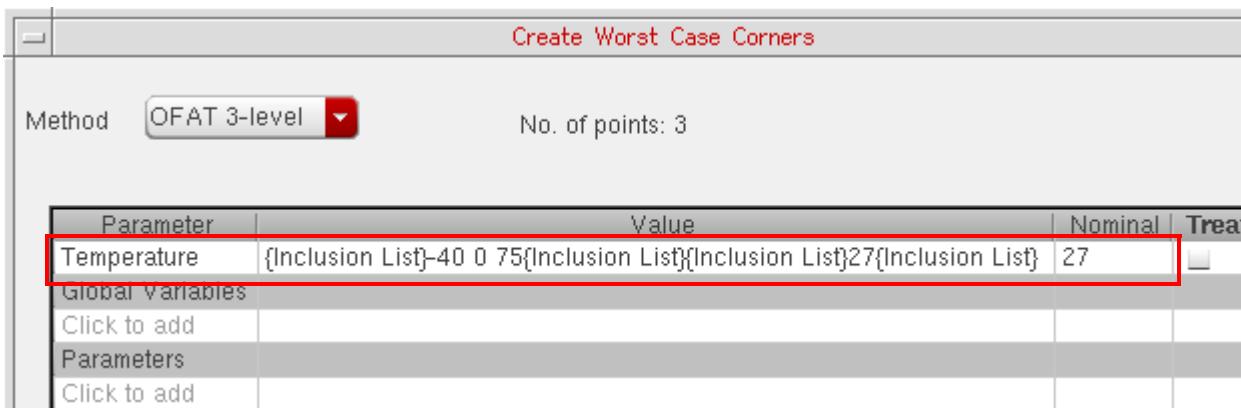
2. In the table given below the *Method* field, specify a range or a set of sweep values for Temperature, if you want to vary temperature.
3. Type in a nominal temperature value in the *Nominal* cell for temperature or double-click in the cell to open the drop-down list.  
All the values specified in the *Value* cell are displayed in the list.
4. Select a nominal temperature value from this list.

**Note:** The nominal value has to be in the given value list. If you type in a nominal temperature value which is not in the specified value list, that value is not accepted by the form.

Alternatively, you can get the value of temperature from the test. For this, right-click in the *Nominal* cell for temperature and choose *Get Value from Test*. Note that if the value taken from the test does not exist in the *Value* cell, the tool automatically creates an inclusion list to add the nominal value to the existing values.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

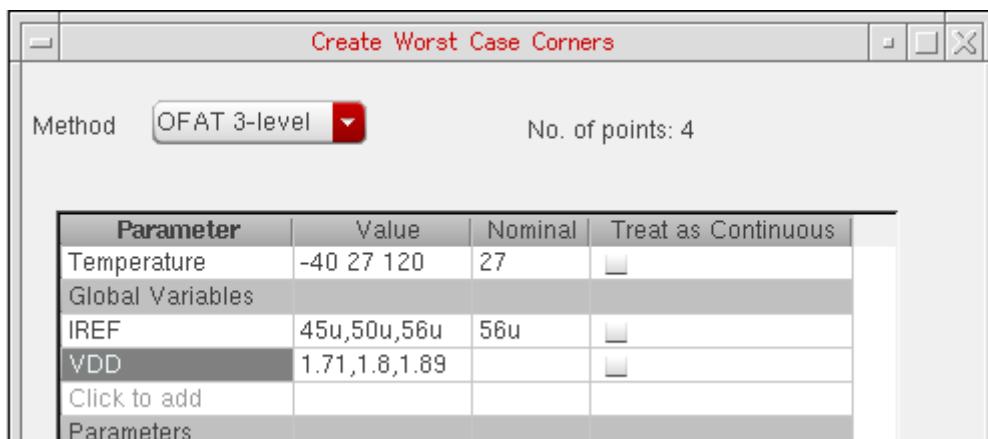


5. In the *Global Variables* field, click *Click to add*.

A drop-down list containing variables is displayed.

6. Select the required variables.

The tool reads the range of values from the ADE XL setup and displays them in the *Value* column. The tool also reads the value of that variable from the test and if that value is one of the values in the *Value* column, displays the same in the *Nominal* cell.



For example, in the above figure, the value for *IREF* in the test is 56u, which is there in the value list. Therefore, the tool automatically fills that value in the *Nominal* cell. However, for *VDD*, the value in the test is 2.0, which is not in the given value list. Therefore, no value is displayed in the *Nominal* column. In this case, you can choose to get the nominal value either from the test, setup state, or from the reference point. For that, right-click in the *Nominal* column and choose an appropriate option. If the value taken from any of these sources does not exist in the *Value* cell, the tool automatically

creates an inclusion list to add the nominal value to the existing set of values in the *Value* cell.

**Note:** If required, you can edit the value list in the *Value* cell. For more details, refer to [Step 9](#) below.

**Note:** As you add more variables and parameters, the tool dynamically calculates the number of points and changes the value in the *No. of points* label. For more details, refer to [Calculation of the Number of Data Points](#).

7. In the *Parameters* section, click on `click` to add.

A drop-down list containing the parameters is displayed.

8. Select the required parameters.

Similar to the global variables, the tool reads the range of values and design value for the selected parameters from the Variables and Parameters Assistant and displays them in the *Value* and the *Nominal* column. Similar to the global variables, if the nominal value is not one of the values in the Value column, the *Nominal* cell is blank. You need to fill in that value.

**Note:** If there is only a single value in the Value column, the same value is filled in the Nominal cell as well. However, if required, you can change the nominal value. In that case, the tool creates an inclusion list with the values from both the *Value* and *Nominal* columns.

9. If required, change the values of variables and parameters. For this, you can double-click in the *Value* column and edit the values by specifying a range or a list of space-separated values. Alternatively, you can select one or more values and right-click to display the shortcut menu and choose:

- Load Values* to load the values defined in the Data View or the Variables and Parameters Assistant.
- Load Min and Max* to load only the minimum and maximum values defined in the Data View or the Variables and Parameters Assistant.
- Set Sweep/Value* to edit the exclusion/inclusion list.

10. If required, change the nominal value for variables or parameters.

The default nominal value is taken from the values of the parameter in the design. To change the nominal value for a variable or parameter, double-click in the *Nominal* cell for the variable or parameter. A list of possible nominal values is displayed. You can either select a value from the list or edit the value. Note that:

- If the variation method is OFAT 3-level, a list of all possible nominal values calculated using the range of values is displayed.
- If the variation method is OFAT Sweep, all sweep values appear in the list. You can select any value from this or type a different value.

**Note:** If you specify a nominal value outside the specified range or the sweep set, the tool does not accept the value and makes the cell blank. A valid nominal value for a parameter is any value from within the range or sweep set corresponding to the parameter.

**11.** Click on `Click to add` in the Models section in the table.

The Add/Edit Model Files form is displayed.

**12.** Browse and select a model file for which you want to vary different sections.

The name of the file appears in the cell.

**13.** Select the check box in the *Value* cell to use the model file for variation.

**14.** Double-click in the *Value* cell.

A list of sections defined in the specified model file appears.

**15.** Select a section that you want to add to the list.

You can select multiple sections that you want to vary for the model file.

**16.** Double-click in the *Nominal* cell and select name of a section that you want to use as a nominal value for the selected model file.

**Note:** It is also possible to specify a model file without selecting any section. In this case, the model file specified in the Create Worst Case Corners form is used instead of the model file specified with the test.

**17.** Similarly, you can select model groups that you want to vary for the simulation.

**18.** Click *Create Corners Automatically* to automatically create worst case corners for the selected spec.

To create worst case corners for a specific spec, click *Create Corners Interactively*. For more information about interactively creating worst case corners, refer to [Creating Worst Case Corners Interactively](#).

**19.** Click *Apply* to save the settings.

**20.** Click *OK* to close the form.

The setup required for creating worst case corners is complete.

## Calculation of the Number of Data Points

Depending on the range of values and nominal value and the value in the *Method* list, the tool calculates the total number of data points for which simulation is to be run and displays the count in the *No. of Points* label to the right of the *Method* field. As you add more variables and parameters, the tool dynamically changes the value of this label.

**Note:** By default, the *Treat as Continuous* check box is deselected and the variables are not treated as continuous. In this case, while creating a worst case corner, the tool uses only those parameter values that were used to run simulation. You can select this check box if you want the tool to create a quadratic model for the parameter and specification combination. In this case, the tool uses interpolation to find the worst value of the parameter for a particular specification.

Consider the variable selection as shown in the example given below:

Parameter	Value	Nominal	Treat as Continuous
Temperature	0 27 100	27	<input type="checkbox"/>
Global Variables			
CAP	800f 900f 1000f	900f	<input checked="" type="checkbox"/>
beta	80.4 90.4 100	80.4	<input type="checkbox"/>
RES	1K 2K 3K	2K	<input checked="" type="checkbox"/>
Click to add			
Parameters			
R0/r	1K 2K 3K	2K	<input checked="" type="checkbox"/>
Click to add			
Models			
Click to add			
Model Group(s)	<modelgroup>		
Click to add			

In this case, while creating a worst case corner, the tool uses either of 0, 27, or 100 as temperature for the worst case scenario. However, for CAP, it first creates a quadratic model and obtains a value for the worst case corner based on interpolation. By interpolation, it can arrive at any value in between 800f and 1000f, including these two values. The worst case corner value obtained after interpolation is more accurate than the value that is used if the variables are not treated continuous.

## Creating Worst Case Corners



The Create Worst Case Corners run does not run with swept variables and parameters. Therefore, either disable sweeps or ensure that all the variables and parameters that are swept in the Variables and Parameters assistant are overridden in the Create Worst Case Corners options form.

To create worst case corners:

- On the *Run* toolbar, click the *Run Simulation* button.

Two forms are displayed, the Corners Setup form and the Sensitivity Analysis–WorstCaseCorners form.

The Corners Setup form displays the worst case corners that are created. For more information, see [Viewing Worst Case Corners](#).

The Sensitivity Analysis–WorstCaseCorners form displays the sensitivity results for the simulation. For more information, see [Working with Sensitivity Data](#).

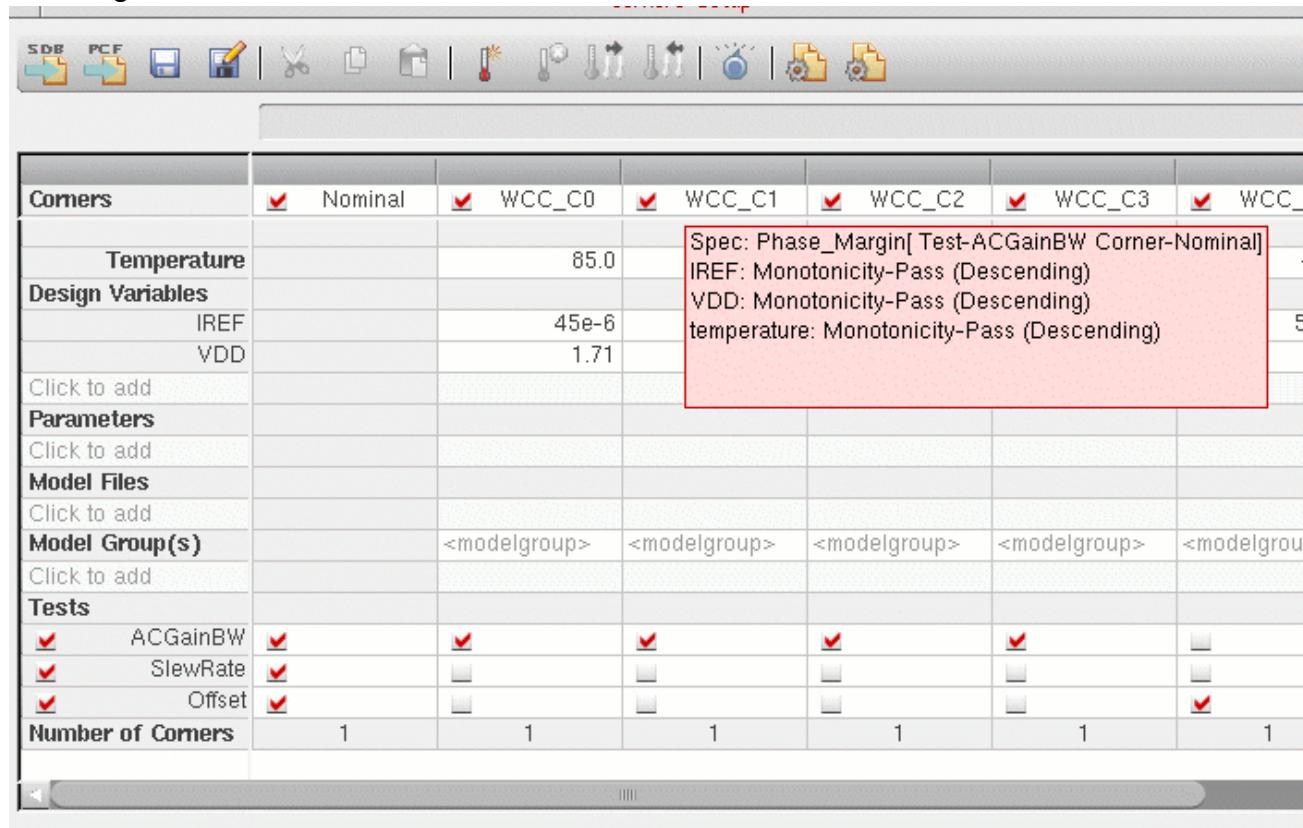
## Viewing Worst Case Corners

Worst case corners are created for every spec and are indicated by the prefix `WCC`. You can view worst case corners by doing one of the following:

- [Using the Corners Setup Form](#)
- [Using the Data View Assistant Pane](#)

## Using the Corners Setup Form

In the Corners Setup form, a tooltip displays the spec and monotonicity of the spec with respect to the corner variable. Point to a worst case corner name to view the tooltip, as shown in the figure below.



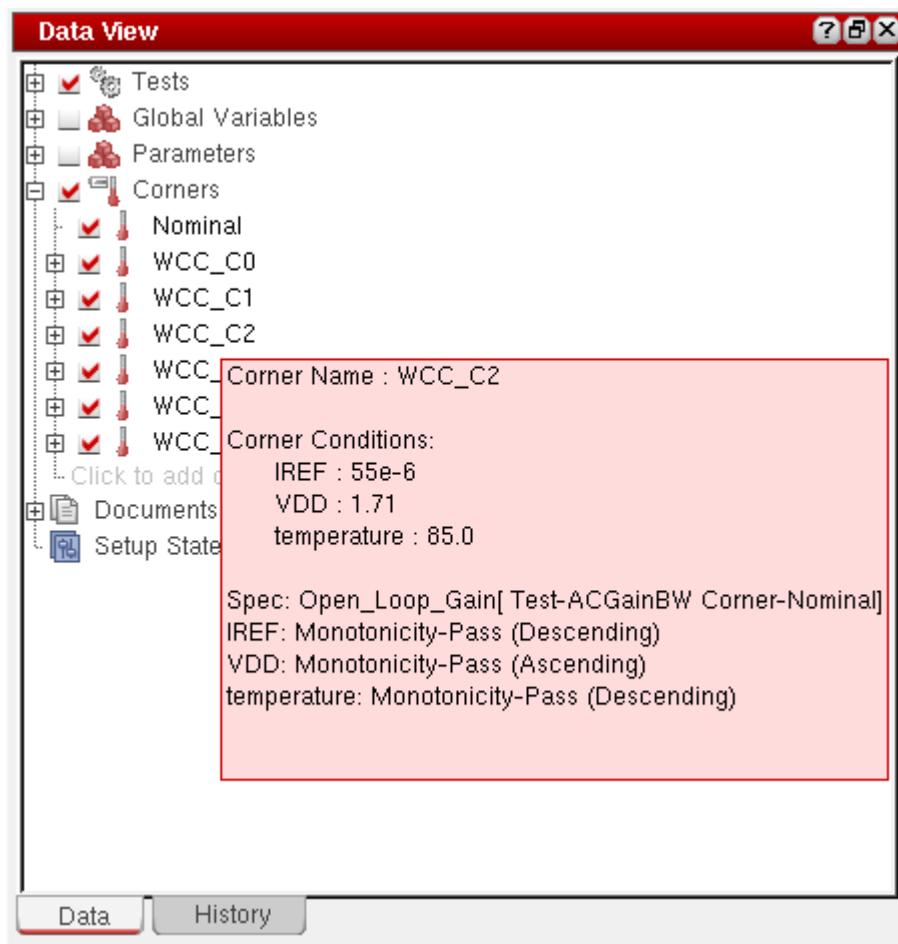
**Note:** *Monotonicity-Fail* indicates that the output is not monotonic with the change in the corner variable.

## Using the Data View Assistant Pane

To view the worst case corners from the Data View assistant pane, follow these steps:

1. In the Data View assistant pane, click the *Data* tab.
2. Expand the *Corners* tree.
3. Point to a worst case corner name, such as `WCC_C2`.

A tooltip, as shown in the figure below, displays the spec and monotonicity of the spec with respect to the corner variable.



## Creating Worst Case Corners Interactively

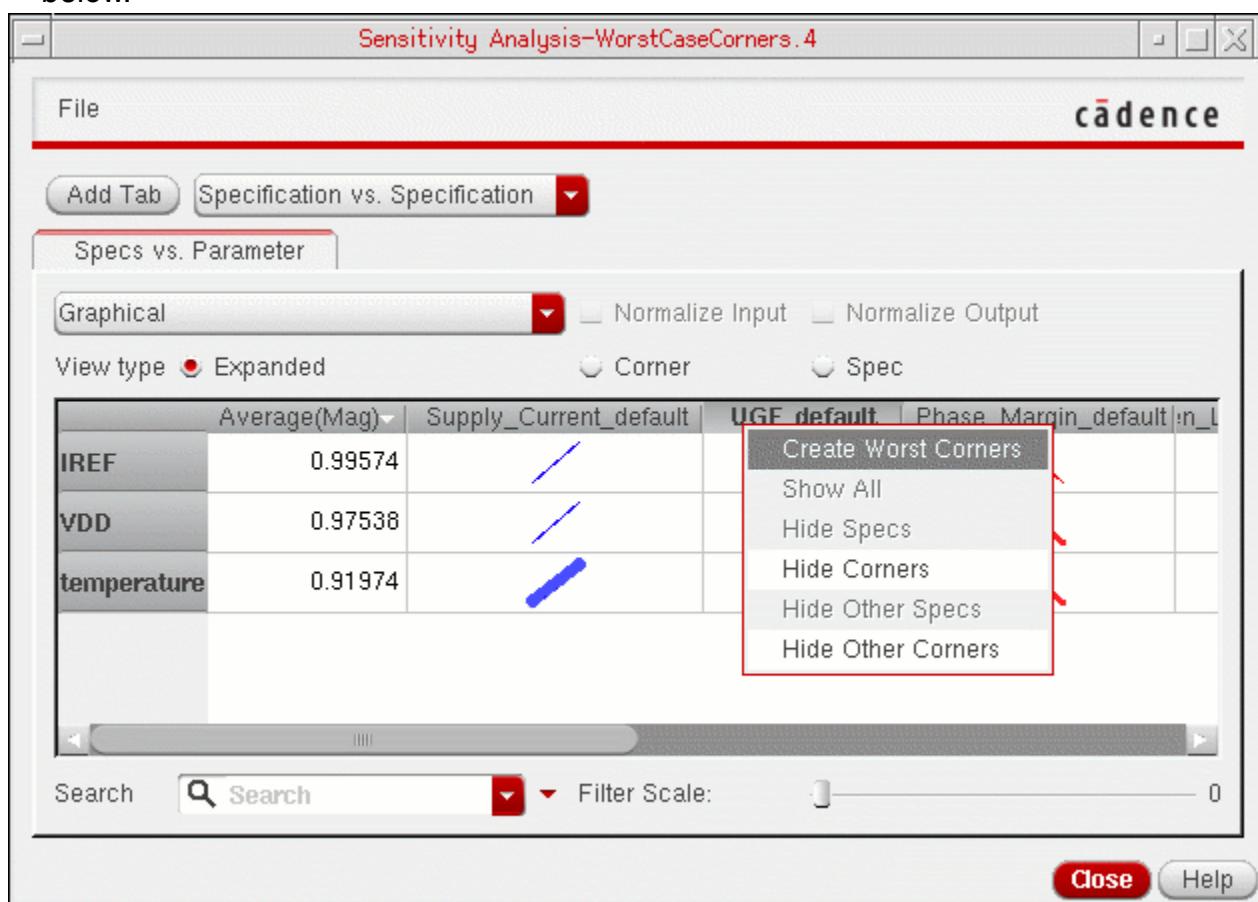
To create worst case corners:

1. Specify the setup for creating worst case corners, as explained in [Specifying the Setup](#).
2. In the Create Worst Case Corners form, select the *Create corners interactively* option, as shown in the figure below.
3. Run the simulation to create worst case corners, as explained in [Creating Worst Case Corners](#).
4. In the Sensitivity Analysis–WorstCaseCorners form, select the spec and corner variable combinations as explained below:

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

- ❑ To create a worst case corner for a single corner variable and a selected spec, select the required cell.
  - ❑ To create worst case corners for a single corner variable and all specs, click the corner variable name. This selects the entire row.
  - ❑ To create worst case corners for all corner variables and a selected spec, click the spec name. This selects the entire column.
  - ❑ To create worst case corners for all corner variables and all specs, hold down the Ctrl key and click each column name (or row name) to select all the cells in the table.
5. Right-click a selected cell and choose *Create Worst Corners*, as shown in the figure below.

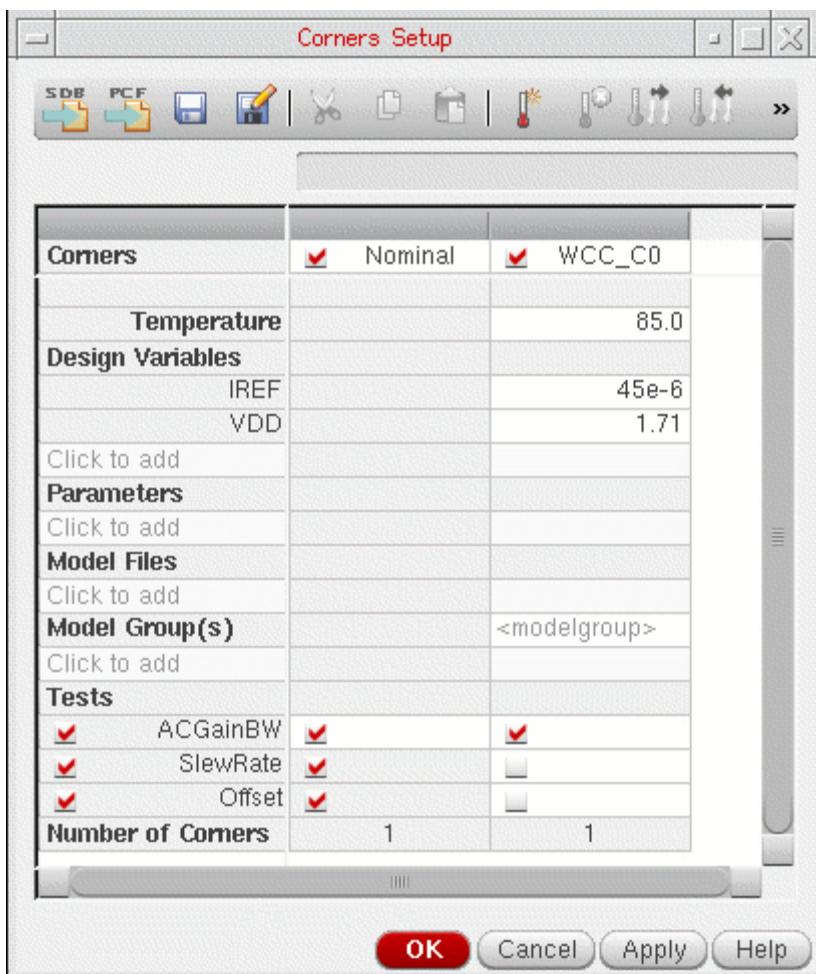


**Note:** In the Graphical view, the row and column name of the selected cell appears in bold. In all other views, the selected cell appears with a gray background.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

The worst case corner is created for the selected combination and is displayed in the Corner Setup form, as shown below.



## **Viewing Mismatch Analysis Results**

You can view correlations between specifications and mismatch parameters from a Monte Carlo run. ADE GXL averages mismatch data over all mismatch parameters on a per-device pass basis.

The average values are calculated as follows:

Average of correlation = Average of absolute values of correlation coefficients for each specification

Average of regression = Average of absolute values of regression coefficients for each specification

Average of normalized regression = Average of absolute values of normalized regression coefficients for each specification

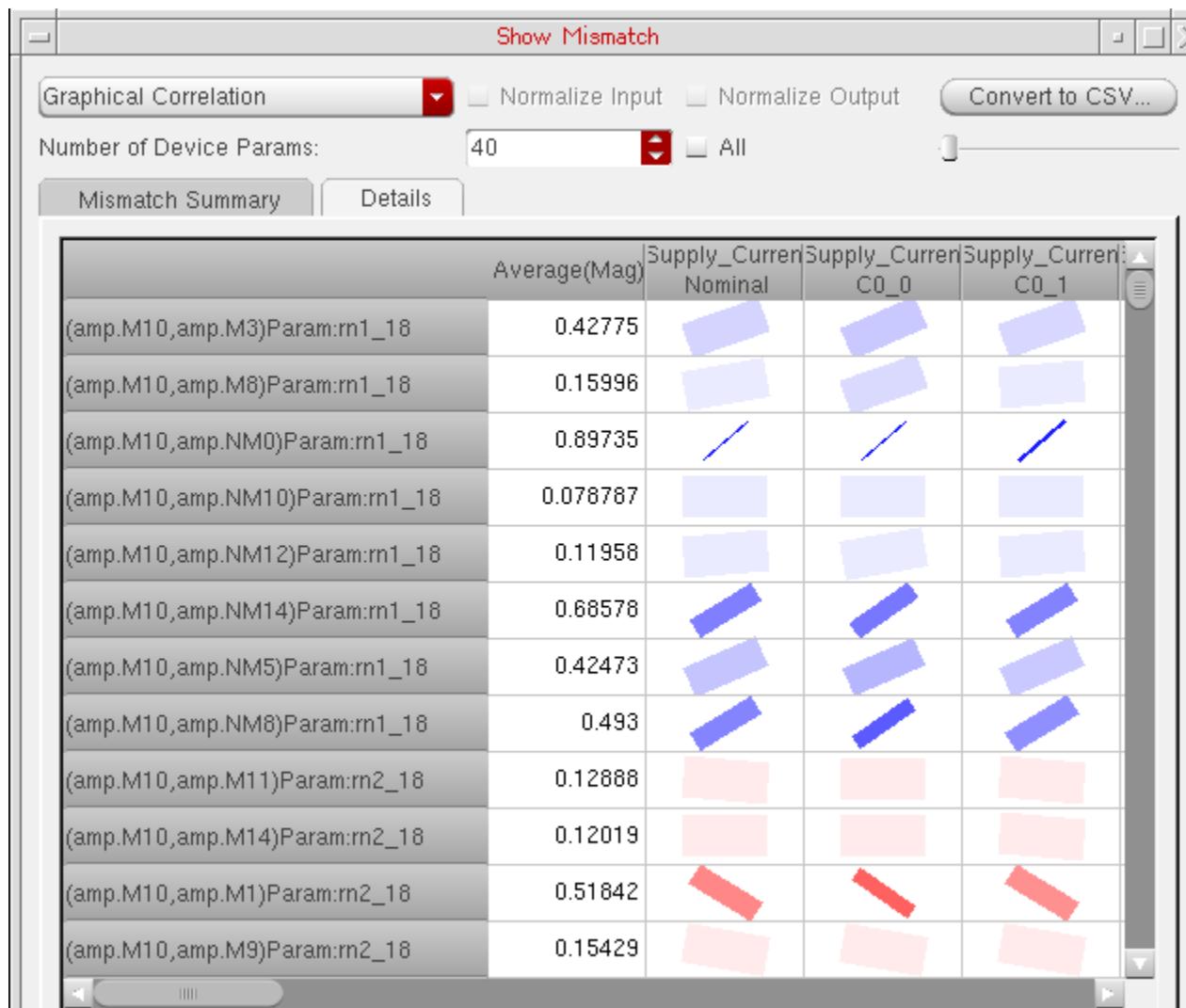
To view the mismatch analysis results for a Monte Carlo run, do one of the following:

- Click the *Mismatch Results*  button on the Results tab.
- In the History tab on the Data View pane, right-click on the history item for a Monte Carlo run and choose *Mismatch Results*.

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

The Show Mismatch form appears.



For each device pair, the Mismatch Summary tab displays average of the absolute values of the mismatch parameters displayed in the Details tab. For example, in the following figure, the Details tab displays the detailed correlation coefficient for all the devices, whereas, the

## Virtuoso Analog Design Environment GXL User Guide

### Sensitivity Analysis

---

Mismatch Summary shows average of the absolute values of correlation coefficient for each device pair.

	Average(Mag)	mn1_ids_default	mn2_ids_default	mn3_ids_default	mn4_ids_default
(mn1,mn2)Param:nvsigma	0.1631	-0.20509	0.37172	-0.049319	-0.000000
(mn1,mn3)Param:nvsigma	0.22991	-0.045167	0.22453	0.50292	-0.000000
(mn1,mn4)Param:nvsigma	0.16033	-0.29833	-0.19728	0.28186	-0.000000
(mn1,mn2)Param:nwsigma	0.34078	-0.72487	0.62806	-0.19815	-0.000000
(mn1,mn3)Param:nwsigma	0.3041	-0.87795	-0.16938	0.29364	-0.000000
(mn1,mn4)Param:nwsigma	0.35309	-0.70542	0.084954	-0.130000	-0.000000
(mn2,mn3)Param:nvsigma	0.000000	-0.000000	-0.000000	-0.000000	-0.000000
(mn2,mn4)Param:nvsigma	0.000000	-0.000000	-0.000000	-0.000000	-0.000000

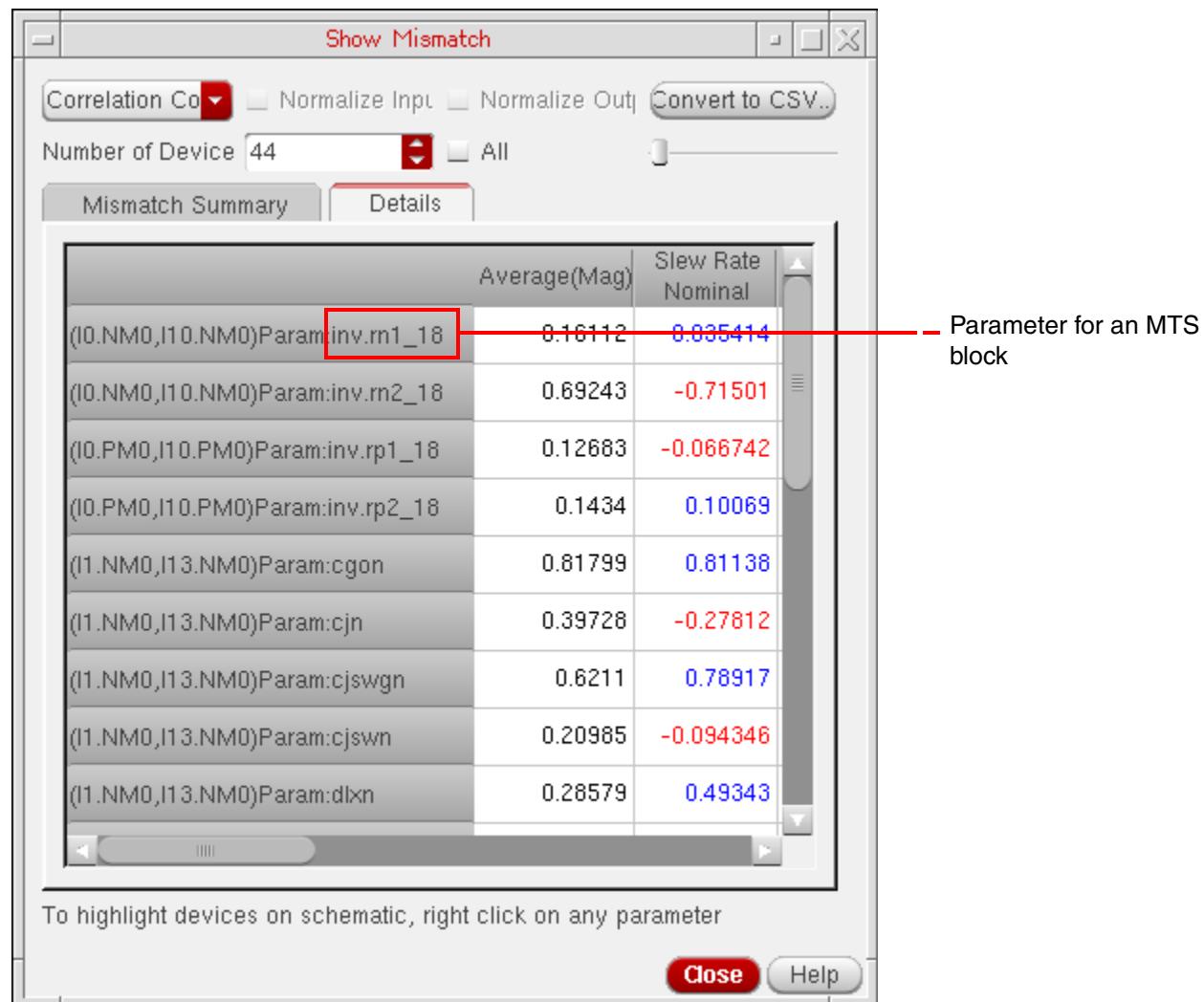
	Average	mn1_ids_default	mn2_ids_default	mn3_ids_default	mn4_ids_default
mn1,mn2	0.25194	0.46498	0.48989	0.1237	-0.000000
mn1,mn3	0.267	0.46156	0.19695	0.3982	-0.000000
mn1,mn4	0.25671	0.50187	0.14112	0.2059	-0.000000
mn2,mn3	0.27828	0.11039	0.57289	0.5637	-0.000000
mn2,mn4	0.20615	0.0079217	0.56086	0.1617	-0.000000
mn3,mn4	0.24369	0.1078	0.36064	0.399	-0.000000
mp1,mp2	0.31572	0.35916	0.32509	0.2998	-0.000000

Note that in the above figure, 0.46498 is the average of the absolute values of correlation coefficient for mn1, mn2 in the Details tab.

## Support for Multi-Technology Simulations in Mismatch Analysis

If you have enabled multi-technology simulations for a test, in the detailed mismatch analysis results, the parameter names for MTS blocks are prefixed with the name of their corresponding block.

For example, in the mismatch data displayed below, the names of the parameter `rn1_18` belongs to an MTS block `inv`. Therefore, the name of the parameter is prefixed with `inv`. Note that in the figure given below, names of all the parameters of block `inv` are prefixed with `inv`.



See the following topics for more information about using the Show Mismatch form:

- [Changing the Type of Results to be Displayed](#) on page 112

- [Changing the Number of Device Parameters](#) on page 112
- [Filtering the Data](#) on page 113
- [Viewing Associated Devices](#) on page 113
- [Saving the Mismatch Data](#) on page 113

## Changing the Type of Results to be Displayed

By default, the mismatch results are displayed in graphical format. You can choose to view the correlation, regression, and normalized regression results generated from mismatch analysis.

To choose the type of results to be displayed, choose one of the following data type from the drop-down list given at the top of the Show Mismatch form:

- *Graphical Correlation*
- *Correlation Thumbnail Plot*
- *Graphical Regression*
- *Correlation Coefficient*
- *Single Variable Linear Regression*
- *Multivariate Linear Regression*

For more information on these data types, see [Changing the Type and Format of Data To View](#).

## Changing the Number of Device Parameters

By default, mismatch analysis results are displayed for all the device parameters in your design. You can change the number of device parameters for which mismatch results are displayed.

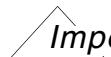
To change the number of device parameters for which mismatch analysis results are displayed:

- Specify the number of device parameters in the *Number of Device Params* field.

**Note:** You must specify a minimum of 2 in the *Number of Device Params* field. The maximum number specified in the *Number of Device Params* field cannot exceed the maximum number of device parameters in your design.

To view mismatch analysis results for all device parameters:

- Select the *All* check box next to the *Number of Device Params* field.

 **Important**

The *Graphical*, *Correlation Coefficient*, and *Single Variable Linear Regression* results share a common value specified for the *Number of Device Params* field. If you change this value for any of these result types, it is changed for all three result types. However, for *Multivariate Linear Regression* results, you can specify a different value for the *Number of Device Params* field. By doing this, you can specify the top *x* number of devices for which you want to see *Multivariate Linear Regression* results.

## Filtering the Data

With the Fit slider bar, you can choose to filter out those relationships that don't show a strong correlation. Depending on how far you move the slider bar, ADE GXL only displays values with a correlation stronger than the location of the slider bar. For example, if you move the slider halfway across the bar, ADE GXL only displays values that have at least 50% of their data points correlating to a regression line.

To filter for values with a stronger correlation:

- Select and drag the slider bar next to the *Number of Device Params* field to the fit level for which you want to filter.

ADE GXL displays only the values that have at least the specified fit.

## Viewing Associated Devices

When you right-click on parameters in the Mismatch matrix (Y-axis) and choose *Highlight on Schematic*, the associated device(s) in the schematic are highlighted.

## Saving the Mismatch Data

You can save the mismatch data to a text file:

1. Click *Convert to CSV*.
2. In the Save As form, specify a path and file name.
3. Click *OK*.

# **Virtuoso Analog Design Environment GXL User Guide**

## Sensitivity Analysis

---

---

## Circuit Optimization

---

In addition to parameterizing a design for sizing and optimization using ADE variables, you can also use a parameterization flow in ADE GXL. To use this flow, you can parameterize a design and set specifications, then perform a local or global optimization and back-annotate the results of the optimization.

This chapter covers the following tasks:

- [Parameterizing the Design](#) on page 116
- [Setting Up Specifications](#) on page 122
- [Running Optimization](#) on page 130
- [Manual Tuning](#) on page 137
- [Sizing Over Corners](#) on page 143
- [Running Feasibility Analysis](#) on page 148
- [Improving the Yield](#) on page 151
- [Creating, Viewing, and Modifying Reference Points](#) on page 157
- [Specifying How Much Optimization Data to Save](#) on page 163

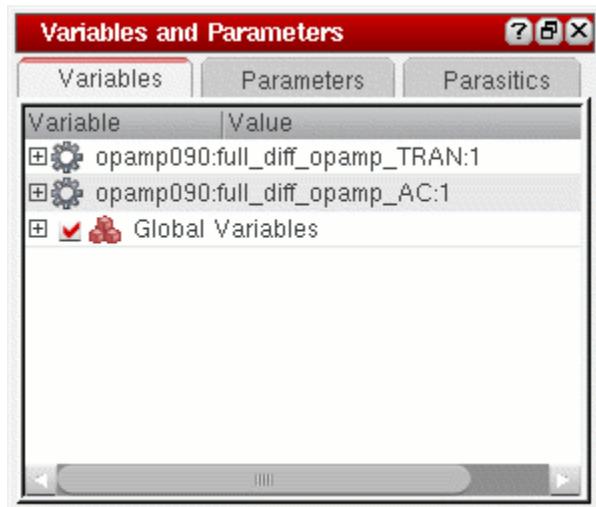
## Parameterizing the Design

When you parameterize the design for optimization, you capture the critical device relationships in the circuit. For example, the input transistors of an opamp must be matched, and the transistors of a current mirror must be ratioed. In order to run optimization, you must define a range of legal values for optimization for each transistor that is the “master” in a matching relationship.

To begin parameterizing the design.

- Choose *Window — Assistants — Variables and Parameters*.

The Variables and Parameters pane appears.



## Matching Devices and Device Properties

You have three options for matching. You can match all device properties for a device, or you can match only specific properties. You can also ratio-match. When you’re finished specifying device relationships, you can define legal values for device properties.

### Matching Devices

To match devices:

1. Select the devices you want to match in the schematic.

The devices are listed on the upper half of the Parameters tab of the Variables and Parameters assistant pane.

2. In the Parameters tab, select the device you want as the “master device.” The master device is the device that all the other selected devices will match.
3. Click the *Match Parameters*  button.

The matched parameters appear on the lower half of the Parameters tab of the Variables and Parameters assistant pane.

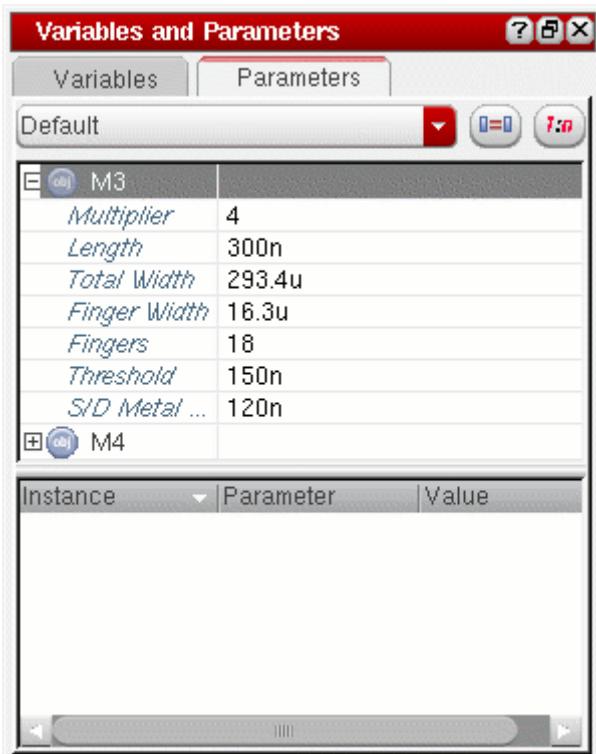
You can now modify the values of matched parameters to vary them during the optimization process.

## Matching Device Properties

To match specific device properties:

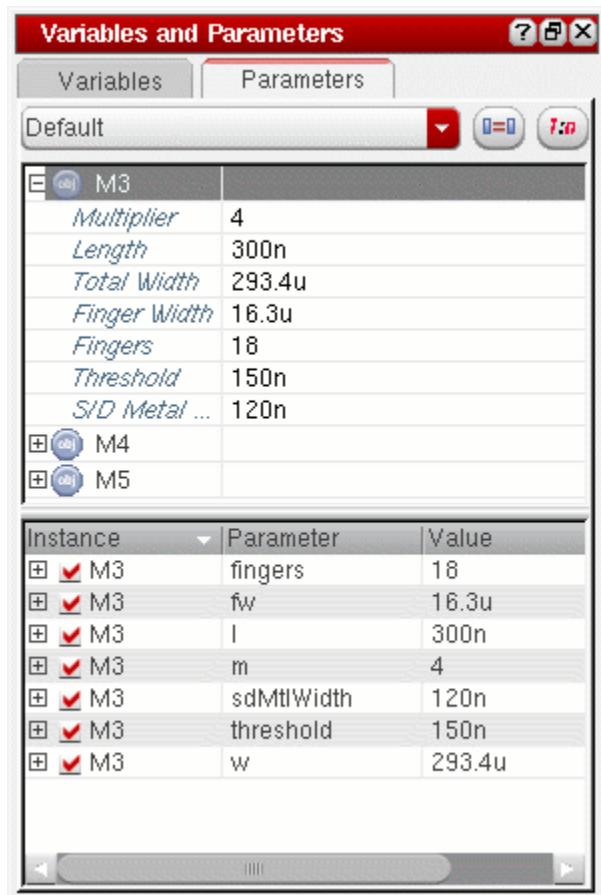
1. In the schematic, click the devices for which you want to match properties.  
As you click, each device and its parameters appear on the upper half of the Parameters tab of the Variables and Parameters assistant pane.
2. Ensure that *Default* is selected in the drop-down list on the upper half of the Parameters tab of the Variables and Parameters assistant pane.
3. Click the + sign next to a device with whose properties you want to match the properties of other devices.

The list of properties for the device appears, as well as the schematic value of each property.



4. Select the property you want to match.
5. Click the *Match Parameters* button.

The matched parameters appear on the lower half of the Parameters tab.



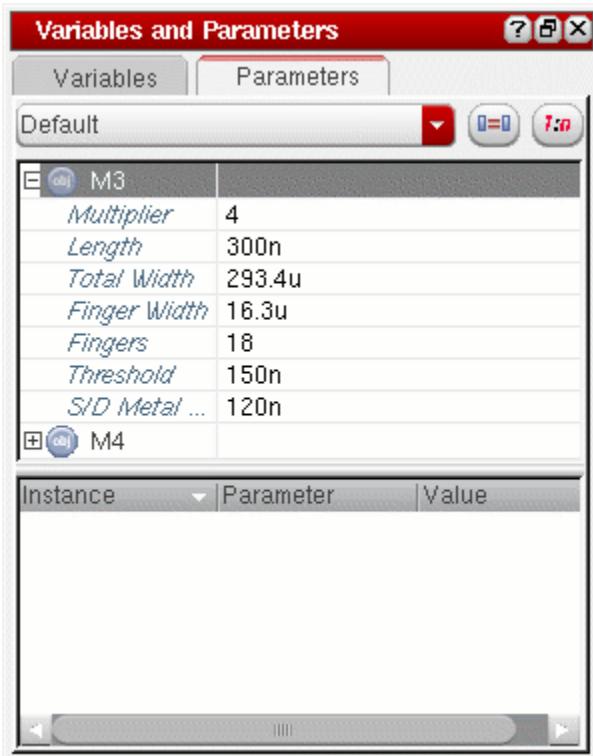
You can now modify the values of matched parameters to vary them during the optimization process.

## Ratio-Matching Device Properties

To ratio-match specific device properties:

1. In the schematic, click the devices for which you want to ratio-match properties.  
As you click, each device and its parameters appear on the upper half of the Parameters tab of the Variables and Parameters assistant pane.
2. Ensure that *Default* is selected in the drop-down list on the upper half of the Parameters tab of the Variables and Parameters assistant pane.

3. Click the + sign next to a device with whose parameters you want to ratio-match the parameters of other devices.



4. Select the parameters you want to ratio-match.
5. Click the *Ratio Matched Parameters*  button.

ADE GXL automatically ratios the parameters based on the lowest value.

The ratio-matched parameters appear on the lower half of the Parameters tab.

You can now modify the values of ratio-matched parameters to vary them during the optimization process.

## Defining Values for Optimization

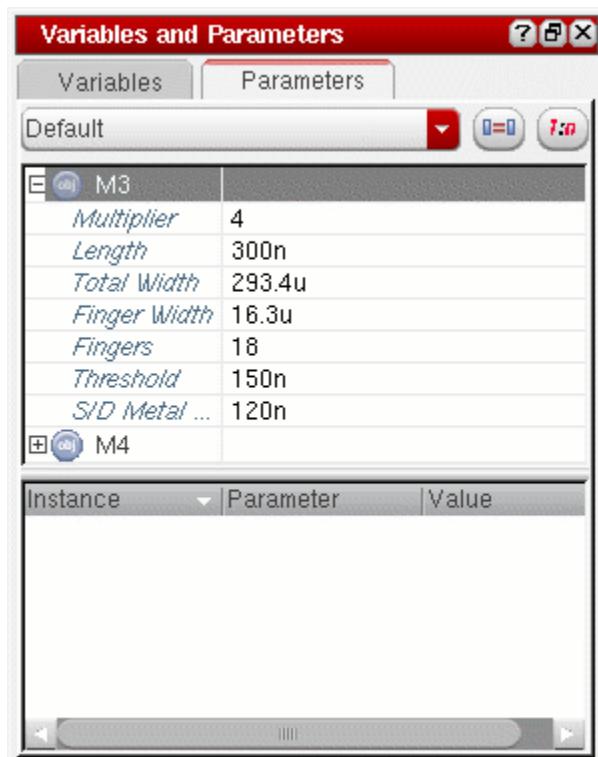
To define a legal range of values for devices:

1. In the schematic, click the devices for which you want to specify values.

As you click, each device and its parameters appear on the upper half of the Parameters tab of the Variables and Parameters assistant pane.

2. Ensure that *Default* is selected in the drop-down list on the upper half of the Parameters tab of the Variables and Parameters assistant pane.
3. Click the + sign next to a device whose parameters you want to modify for optimization.

The list of parameters for the device appears, as well as the schematic value of each property.



4. Click on the value for the parameters you want to modify and type a range or list of values. Specify ranges in the format MIN : STEP : MAX. You can separate a list of values with either commas or spaces.

The modified parameters appear on the lower half of the Parameters tab.

## Setting Up Specifications

When you set specifications on outputs, you establish the performance specifications that ADE GXL must meet during synthesis. These specifications are used as benchmarks for candidate circuit quality during optimization.

You can set up general specifications, area constraint specifications and operating region specifications. For more information, see the following topics:

- [Specifying General Specifications](#) on page 122
- [Specifying Area Constraint Specifications](#) on page 125
- [Running Optimization](#) on page 130

### Specifying General Specifications

For each specification, you specify an objective, or performance constraint, then specify a target value for that specification.

#### Objective Types

There are five types of objectives:

- open-ended
- close-ended
- range
- tolerance
- info

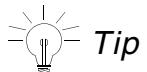
An open-ended specification is when you choose either minimize (min) or maximize (max), then specify an ideal value. For an open-ended specification, ADE GXL tries to continually improve upon the ideal value. So, if the ideal value for the *ugf* specification is maximize to 1e+8, 3e+8 is better than 2e+8.

A close-ended specification is when you specify a target value, then specify that the resulting specification must be greater than (>) or less than (<) that target value. For a close-ended specification, ADE GXL attempts to meet the target value, and exceeding the target is no better than meeting the target. So, if the target value for *ugf* is >1e+8, 3e+8 is no better than 2e+8.

You can also specify a range specification, in which you specify both an upper and a lower boundary for the target value. For a tolerance (tol) specification, you specify a target value and an allowable percentage deviation.

You can specify an info specification when you want to see the value of a measurement, but don't want that specification to affect sizing.

Design specifications might include specifying an open loop gain maximized with an acceptable value of 60dB, or a settling time < 15ns.



*Tip*  
You can also specify area goals. For more information, see [“Specifying Area Constraint Specifications”](#) on page 125.

## Specification Weights

ADE GXL supports weighting on specifications, that is, the ability to put particular emphasis on one or more specifications. You may use weighting if you have certain important specifications, and it is imperative that ADE GXL meet those specifications, even at the expense of other specifications. Or, if you have run ADE GXL without any specification weights, and you have one particular specification that is not being met, you can put emphasis on that specification in order to ensure that ADE GXL meets it.

To set specification weighting, you specify a positive integer as the “weight.” ADE GXL multiplies the cost function for that specification by the integer specified for the weight. Because this specification is now “x” times more important than other specifications, it changes the way ADE GXL searches the design space. ADE GXL will put particular stress on meeting that specification, even at the expense of specifications weighted at “1.” This discrepancy is especially true of specifications that are minimized or maximized, because ADE GXL is already working harder to optimize the value for that specification.

In general, ADE GXL is able to find a working solution without specification weighting. As a result, Cadence advises that you run ADE GXL first with all specifications weights set to “1.” If ADE GXL is repeatedly unable to meet a particular specification, you can put more emphasis on meeting that specification using specification weighting.

## Setting Up Specifications

To set up specifications:

1. In the Outputs Setup tab, double-click on the *Spec* column in the row for the output for which you want to set up specifications.

A drop-down list of objectives appears.

Type	Expression/Signal/File	Plot	Save	Spec	Weight
signal	/outdiff	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
signal	/OUTN	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
signal	/OUTP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
expr	dB20(value(VF("/outdiff") 1))	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	maximize ▾ 7.5	1
expr	abs(IDC("/V0/PLUS"))	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	none	m
signal	/NVCM	<input type="checkbox"/>	<input checked="" type="checkbox"/>	minimize	
expr	abs((VDC("/inn") - VDC("/in..."))	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	maximize ▾ n	1
signal	/V0/PLUS	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<	
signal	/inn	<input type="checkbox"/>	<input checked="" type="checkbox"/>	>	
signal	/inp	<input type="checkbox"/>	<input checked="" type="checkbox"/>	range	
expr	gainBwProd(VF("/outdiff"))	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	tol	
signal	/outdiff	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	info	
signal	/OUTN	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
signal	/OUTP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
expr	(settlingTime(VT("/outdiff") 0 t...)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	< 10n	1
expr	slewRate(VT("/outdiff") 0 t ym...)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	> 200M	1

- Choose an objective from the drop-down list.

For more information on the types of objectives, see [Objective Types](#).

- Select the field next to the drop-down list.
- Enter a target value in the field.

Type	Expression/Signal/File	Plot	Save	Spec	Weight
signal	/outdiff	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
signal	/OUTN	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
signal	/OUTP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
expr	dB20(value(VF("/outdiff") 1))	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	maximize ▾ 7.5	1
expr	abs(IDC("/V0/PLUS"))	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	< 10m	
signal	/NVCM	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

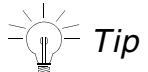
- (Optional) Specify a weight for the specification in the Weight column.

For more information on weights, see [Specification Weights](#).

- Repeat [step 1](#) through [step 5](#) for all outputs for which you want to set specifications.

## Specifying Area Constraint Specifications

You can specify area constraint specifications for use in optimization runs. For each device, you can either manually specify an area formula, or use a CDF parameter to set default area formulas for devices. For more information about using a CDF parameter to set default area formulas, see [Setting Default Area Formulas](#) on page 125.



*Tip*  
You can add multiple area constraint specifications for a test if you want to set different specifications for some devices.

For more information, see the following topics:

- [Setting Default Area Formulas](#) on page 125
- [Setting Up Area Constraint Specifications](#) on page 126
- [Modifying an Area Constraint Specification](#) on page 128
- [Copying a Device Formula](#) on page 128
- [Including Devices in the Area Constraint Specification](#) on page 128
- [Excluding Devices from the Area Constraint Specification](#) on page 129
- [Deleting Devices with Area Formulas](#) on page 129

## Setting Default Area Formulas

You can specify default area formulas for devices using the `AreaFormula` CDF parameter.

If you want to use a different CDF parameter name for specifying default area formulas, specify the parameter name using the `defAreaProp` environment variable in one of the following ways:

- In the CIW, enter the following:  

```
envSetVal("adexl.gui" "defAreaProp" 'string "<myParamName>"')
```
- In your `.cdsenv` file, enter the following:  

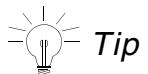
```
adexl.gui defAreaProp string "<myParamName>"'
```

For more information about CDF parameters, see the [Component Description Format User Guide](#).

## Setting Up Area Constraint Specifications

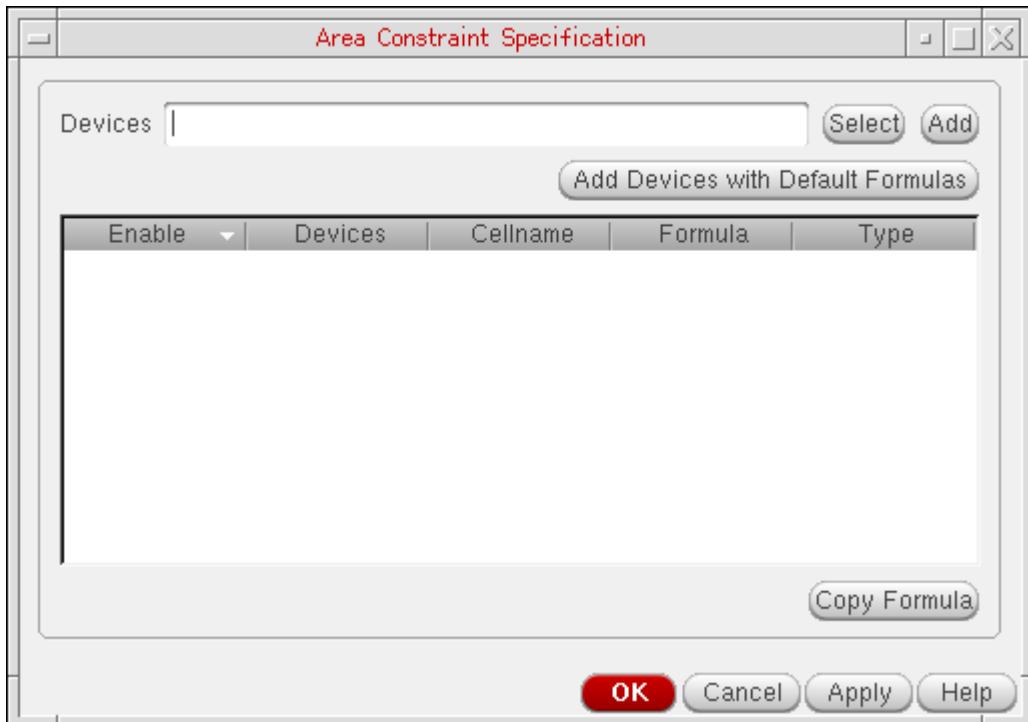
To set up area constraint specifications, do the following:

1. On Outputs Setup tab, click the *Add new output*  button.
2. In the drop-down list, select a test and choose *Area Specification*.



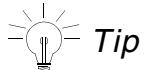
You can also right-click on a test name in the Outputs Setup tab and choose *Add Area Specification* to set up area constraint specifications for that test.

The Area Constraint Specification form appears.



3. Specify a device name for which you want to add an area formula:
  - Select a device in the schematic. To do this, click the *Select* button to display the schematic window. Select the device in the schematic window and press the *Esc* key. The device name is displayed in the *Devices* field.
  - Enter the device name in the *Devices* field.
4. Click *Add* to add the device to the list.
5. Select the *Formula* field for the device and enter an area formula.

You must enter an arithmetic combination of valid CDF parameters that are specified for the device.

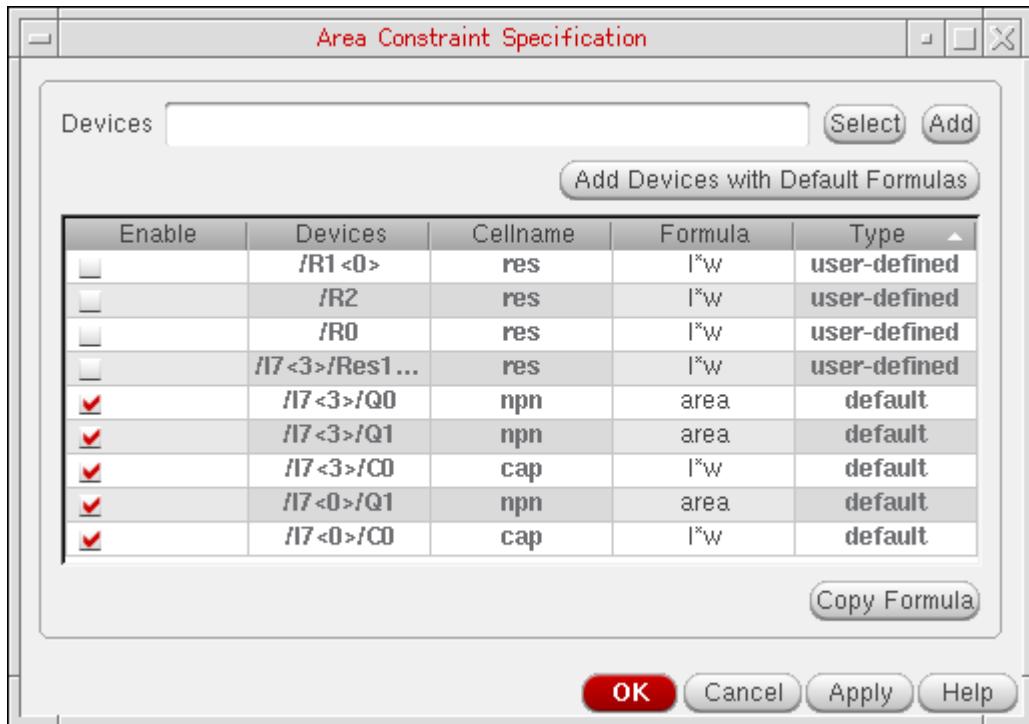


*Tip*

You can use the [Copy Formula](#) button to copy area formulas from one device to another.

6. Repeat [step 3](#) through [step 5](#) for any other devices you want to include in the area constraint specification.
7. Click the *Add Devices with Default Formulas* check box to add all instances in the design for which default formulas are specified using the `AreaFormula` CDF parameter. For more information, see [Setting Default Area Formulas](#) on page 125.

**Note:** You can modify any default formula by selecting the *Formula* field and typing a new area formula.

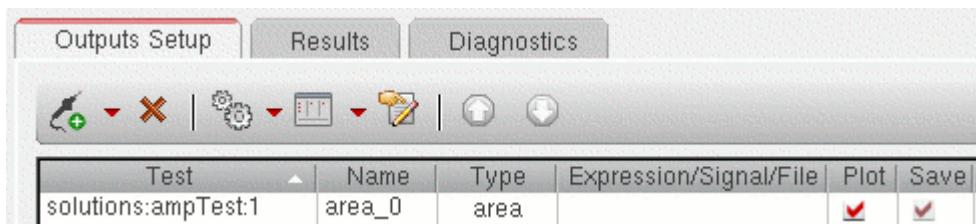


8. To include a device in the area constraint specification, select the *Enable* check box for the device. De-select the *Enable* check box for the devices you do not want to include in the area constraint specification.

**Note:** All devices with default area formulas are included by default, while any devices for which you manually entered area formulas are not.

**9.** Click *OK*.

The area constraint specification is displayed in the Outputs Setup tab.



The area constraint specification is assigned the name `area_seqNum`, where `seqNum` is 0 (zero) for the first area constraint specification you add. You can double-click on the *Name* field and modify the name.

**10.** Set up specifications for the area constraint specification. For more information, see [Setting Up Specifications](#) on page 123.

### Modifying an Area Constraint Specification

To modify an existing area constraint specification:

1. In the Outputs Setup tab, double-click on the *Expression/Signal/File* field for the area constraint specification.

The Area Constraint Specification form appears.

2. Modify the area constraint specification as required.

### Copying a Device Formula

To copy the area formula from one device to another:

1. Select the device from which you are copying the formula.
2. Control-click on the device to which you want to copy the formula.
3. Click *Copy Formula*.

### Including Devices in the Area Constraint Specification

- To include a device in the area constraint specification, select the *Enable* check box.

To include more than one device at once:

1. Select the devices you want to include.
2. Right-click and choose *Enable*.



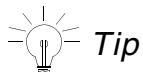
You can enable all devices by right-clicking and choosing *Enable All*.

### **Excluding Devices from the Area Constraint Specification**

- To exclude a device from the area constraint specification, deselect the *Enable* check box.

To exclude more than one device at once:

1. Select the devices you want to exclude.
2. Right-click and choose *Disable*.



You can disable all devices by right-clicking and choosing *Disable All*.

### **Deleting Devices with Area Formulas**

To delete devices and their area formulas:

1. Select one or more devices.
2. Right-click and choose *Delete*.

## Running Optimization

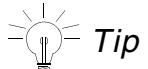
You have three options for optimization:

Local optimization runs the schematic design, then deterministically improves that design. Local optimization is useful if:

- You have a starting design that you want to improve.
- A design meets specifications at the nominal corner but not across all corners.
- Models in a PDK have changed just enough to cause the design to no longer meet all specifications.

Note that local optimization only searches the design space around the specified point, so ADE GXL may not locate the best point possible to meet the design specifications. In addition, local optimization does not guarantee small perturbations to your variable values.

Unlike local optimization, global optimization does not require a reference point. It efficiently searches over all of the variables defined to find a good solution for your design.



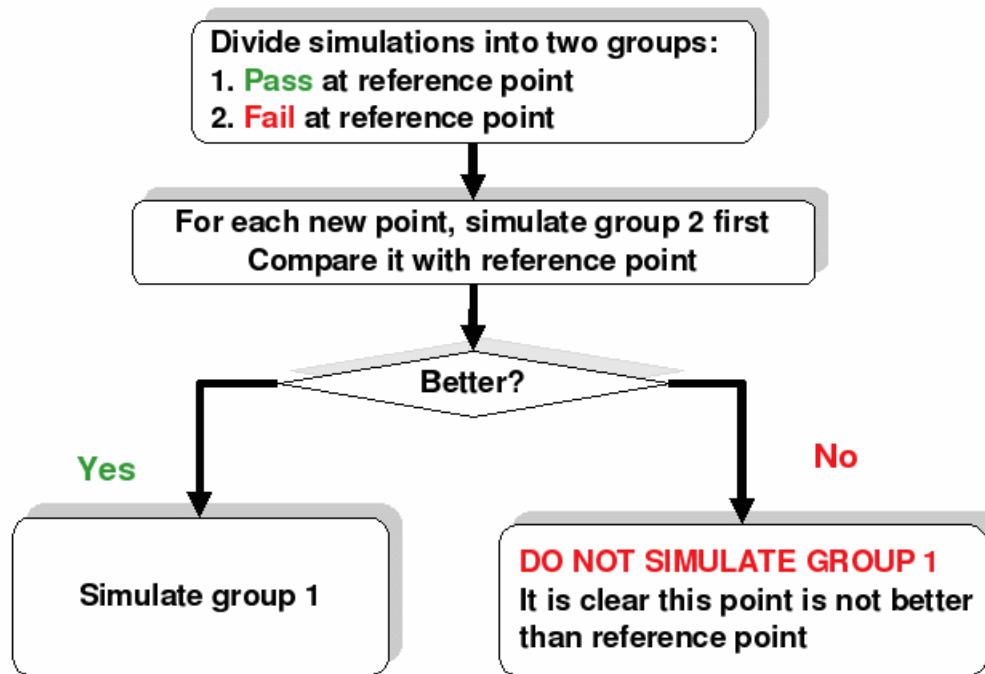
### *Tip*

If you have a large number of corners, you may want to run Size Over Corners instead of running global or local optimization. For more information, see [Sizing Over Corners](#) on page 143.

## Understanding Conditional Evaluation

As ADE GXL is running the simulations for each design point, it may encounter a point where the first simulation run is not as good as the current best design point. In this case, ADE GXL will not run any remaining simulations in order to more quickly and effectively reach a solution. This type of evaluation is called conditional evaluation, and may result in only partially-evaluated points.

When sizing a design with conditional evaluation set, ADE GXL follows the following procedure:



The “No” case results in a partially-evaluated point.

## Running a Local Optimization

You can perform local optimization by using any of the four algorithms: *Brent-Powell*, *Hooke-Jeeves*, *BFGS*, and *Conjugate Gradient*. You can consider the following points while making a choice of an algorithm to be used to run local optimization:

- The *BFGS* and *Conjugate Gradient* algorithms are gradient based algorithms. These methods do well when gradient information is available, that is, when specification measurements return more than two or three states and they do not result in evaluation errors.  
  
 BFGS uses second order Hessian Matrix to search local optimum, which is much more efficient than Conjugate Gradient if the performance space is closer to quadratic. Therefore, it is recommended to use the BFGS algorithm in cases where performance gradient can be calculated and design variables are naturally continuous.
- The *Brent-Powell* algorithm searches in a fine grid around the starting point. Use the *Brent-Powell* option if your reference point is already close (e.g. only a few

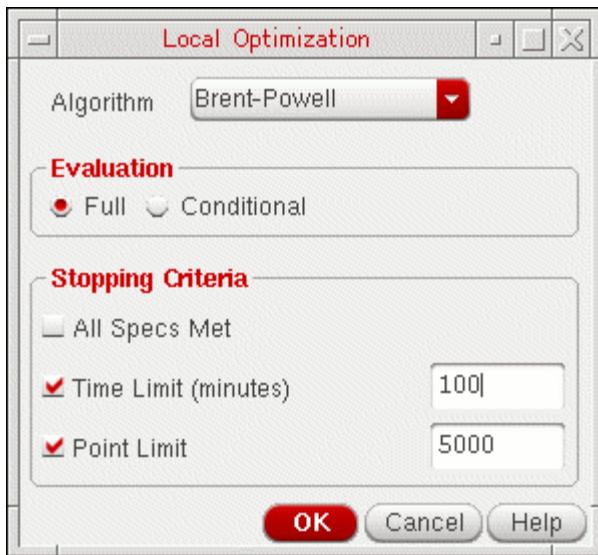
specifications not met). With this type, ADE GXL should find the local minimum around the selected point, but it may also quit once that local minimum is found and then miss a more optimal solution in the local design space. Because it only runs one point at a time, ADE GXL is slow with this option, and it runs the least number of points on average.

- The type *Hooke-Jeeves* algorithm searches in bigger steps, but it can miss a local minimum that might have been found by the *fine* option. Use the *Hooke-Jeeves* option if your time is limited and the starting point is not as close. With this option, ADE GXL is less likely to quit prematurely, but it also may miss a local minimum that the *Brent-Powell* option would have found. With the *Hooke-Jeeves* option, ADE GXL runs more points, but it is faster on multiple machines than when running with the *Brent-Powell* option because of parallel points.

To run local optimization:

1. From the *Select a Run Mode* drop-down list on the Run toolbar, choose *Local Optimization*.
2. Click *Simulation Options*  to specify optimization options.

The Local Optimization Options form appears.



3. From the *Algorithm* drop-down list, select the algorithm for local optimization.

The following algorithms are supported:

- Conjugate Gradient
- Brent-Powell
- Hooke-Jeeves

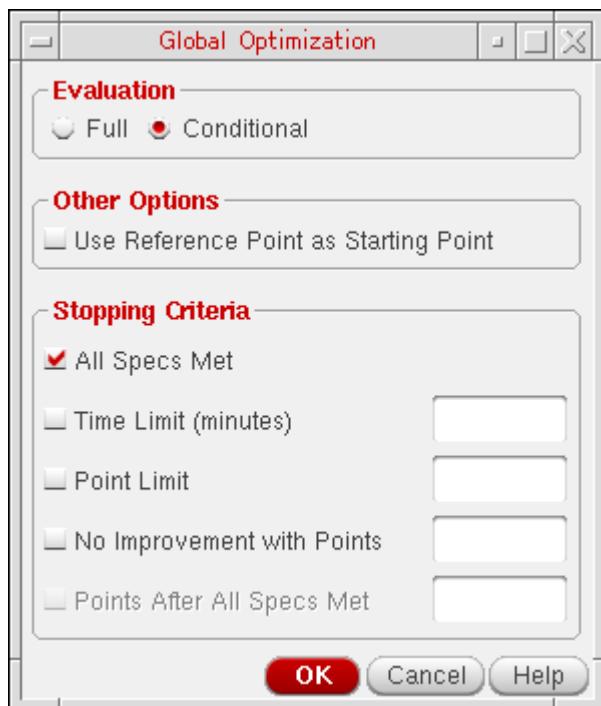
- BFGS
4. Select an evaluation type by selecting one of the following radio buttons:
- Full
  - Conditional
- For more information on conditional evaluation, see [Understanding Conditional Evaluation](#) on page 130.
5. If desired, select criteria for the length of time local optimization should run.
- You can select one or more of the following:
- To run only until all goals are met, select the *All Specs Met* check box.
  - To set a time limit for the run, select the *Time Limit* check box and enter a value in minutes.
  - To set a limit in the number of points run, select the *Point Limit* check box and enter the number of points.
6. Click *OK*.
7. Click the *Run Simulation*  button on the Run toolbar to optimize the circuit.

## Running a Global Optimization

To run a global optimization:

1. From the *Select a Run Mode* drop-down list on the Run toolbar, choose *Global Optimization*.
2. Click *Simulation Options*  to specify optimization options.

The Global Optimization Options form appears.



3. Select an evaluation type by selecting one of the following radio buttons:
  - Full
  - ConditionalFor more information on conditional evaluation, see [Viewing the Variable Data from Optimization Results](#).
4. (Optional) Select the *Use Reference Point as Starting Point* check box if you have created a [reference point](#) and want to use that point as the starting point for the run.  
**Note:** You must have a [reference point](#) available to utilize this option.

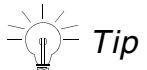
5. If you have created a schematic point or a reference point, and you want to use that point as a starting place for sizing, select the *Use Reference Point as Starting Point* check box.

You must have a reference point available to utilize this option.

6. If desired, select criteria for the length of time local optimization should run.

You can select one or more of the following:

- To run only until all goals are met, select the *All Specs Met* check box.
- To set a time limit for the run, select the *Time Limit* check box and enter a value in minutes.
- To set a limit in the number of points run, select the *Point Limit* check box and enter the number of points.
- To stop sizing when no improvement is seen for a certain number of points, select the *No Improvement with Points* check box and enter the number of points.
- To continue exploring the design space for a better solution even all specifications are met, select the *Points After All Specs Met* check box and enter the number of points to explore.



You cannot select both the *All Specs Met* and *Points After All Specs Met* check boxes, as these two options are mutually exclusive.

7. Click *OK*.

8. Click the *Run Simulation*  button on the Run toolbar to optimize the circuit.

## Viewing the Variable Data from Optimization Results

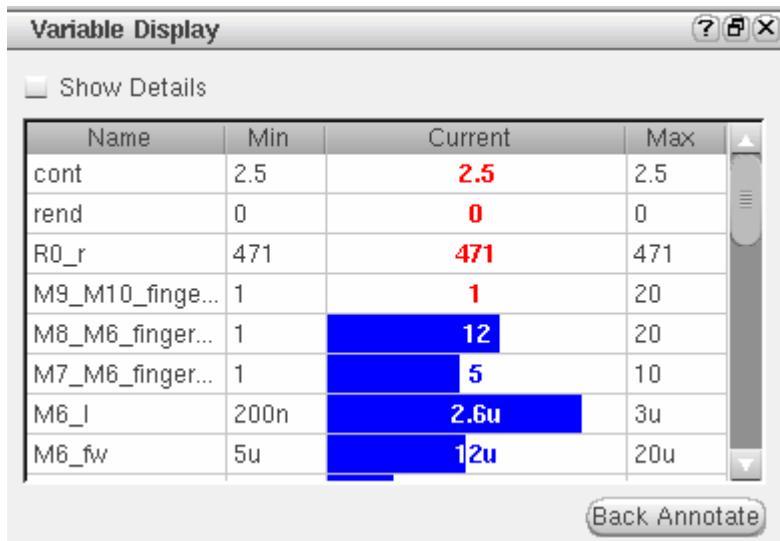
The Variable Display Assistant displays, for a selected design point, the minimum and maximum values for each variable. During optimization, this assistant shows the values for the current best point found, while after optimization, the assistant displays the values for the best point found during the optimization process.

In addition, ADE GXL also displays the current variable values for the selected point, and, through a status bar, shows how far from the minimum and maximum values the current value is. A min or max value in red indicates that the current value is pushing the minimum or maximum and may need adjustment for synthesis to be completely successful.

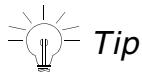
To display the Variable Display Assistant:

- Choose *Window – Assistants – Variable Display*.

The Variable Display Assistant appears.



Name	Min	Current	Max
cont	2.5	2.5	2.5
rend	0	0	0
R0_r	471	471	471
M9_M10_finge...	1	1	20
M8_M6_finger...	1	12	20
M7_M6_finger...	1	5	10
M6_I	200n	2.6u	3u
M6_fw	5u	12u	20u



*Tip*  
You can view more details about the variable, including the lib, cell, and view, by selecting the *More Details* check box.

## Manual Tuning

While parameterizing a design, you can tune your design by varying the values of parameters, running multiple simulations and then comparing results. You can do this by using the Manual Tuning run mode.

Some of the important points related to the Manual Tuning run mode are as follows:

- You run the Single Run, Sweeps and Corners, Monte Carlo, Global Optimization, and Local Optimization run modes multiple times.  
Currently, the Manual Tuning run mode is not supported in the Improve Yield, Sensitivity Analysis, High Yield Estimation, Create Worst Case Corners, Size Over Corners run modes.
- Every time you tune the design parameters and run a simulation, results are appended to the same history checkpoint. This is as compared to the other run modes in which the results of every simulation run is saved in a new checkpoint.
- You can compare the results data in a single table and plot specifications across design points run in different simulations. Therefore, manual tuning, when used with design parameterization, helps you tune your design without changing the schematic.

To run manual tuning:

1. In the adexl view of your design, from the *Select a Run Mode* drop-down list on the Run toolbar, choose *Manual Tuning*.

2. Click the *Run Simulation*  button on the Run toolbar.

**Note:** The color of the *Run Simulation* button changes to yellow  . This indicates that the Manual Tuning run mode has started.

Now, you can tune or vary the values of design parameters and run multiple simulations to obtain the desired results.

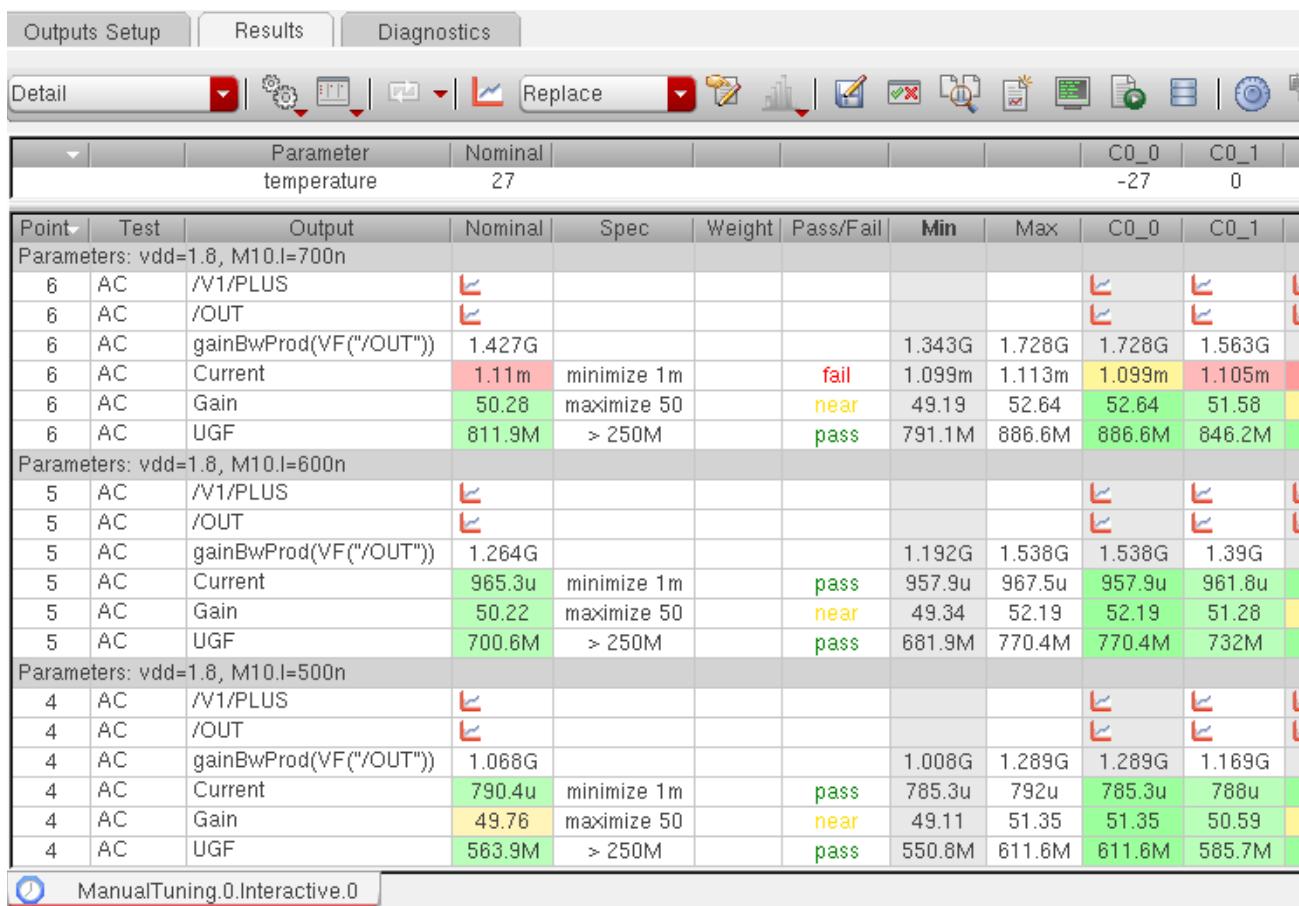
3. In the Data View or the Variables and Parameters assistant, specify values for the design parameters.
4. From the *Select a Run Mode* drop-down list on the Run toolbar, choose *Single Run, Sweeps and Corners*.
5. Click the *Run Simulation*  button on the Run toolbar to start the *Single Run, Sweeps and Corners* run.

The simulation results are displayed on the *Results* tab, as shown below.

# Virtuoso Analog Design Environment GXL User Guide

## Circuit Optimization

---



**Note:** The default name of the Results tab indicates the interactive run number in the Manual Tuning run mode. For example, in the figure shown above, the first single, sweep, corner run in the manual tuning run mode is named as ManualTuning.0.Interactive.0.

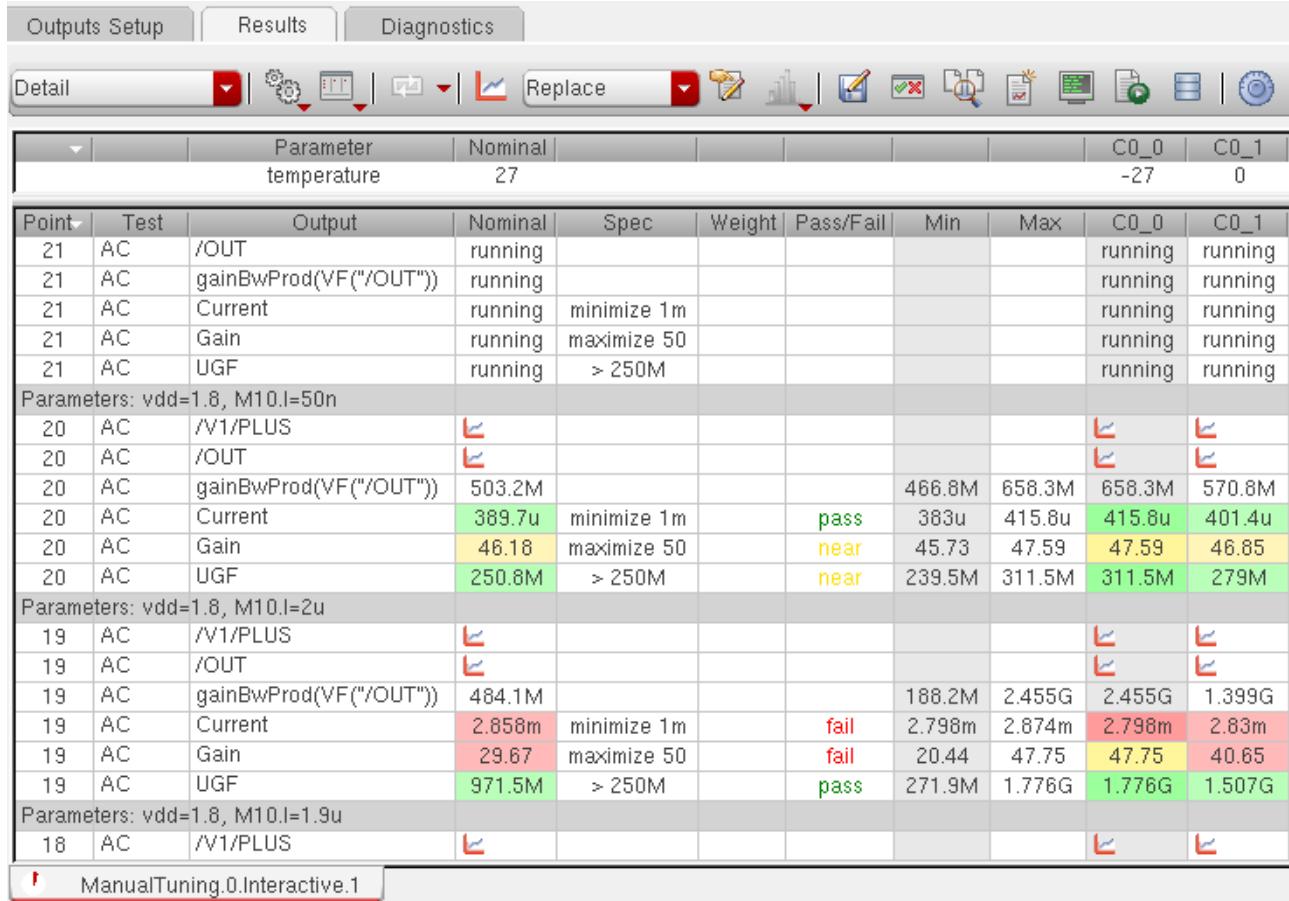
- If required, change the value of parameters to tune your design and again click the *Run Simulation* button on the Run toolbar to start a new *Single Run, Sweeps and Corners* simulation run.

**Note:** The results of the subsequent simulation runs are appended on top of the Results tab already open for the previous run, as shown below.

# Virtuoso Analog Design Environment GXL User Guide

## Circuit Optimization

---



The screenshot shows the Virtuoso Analog Design Environment GXL User Guide interface. The top menu bar includes 'Outputs Setup', 'Results' (which is currently selected), and 'Diagnostics'. Below the menu is a toolbar with various icons for simulation, analysis, and reporting. The main area displays a table of simulation results. The table has columns for Point, Test, Output, Nominal, Spec, Weight, Pass/Fail, Min, Max, and two columns labeled C0\_0 and C0\_1. The table is organized into sections based on parameter values:

- Parameters: vdd=1.8, M10.l=50n**

Point	Test	Output	Nominal	Spec	Weight	Pass/Fail	Min	Max	C0_0	C0_1
21	AC	/OUT	running						running	running
21	AC	gainBwProd(VF("/OUT"))	running						running	running
21	AC	Current	running	minimize 1m					running	running
21	AC	Gain	running	maximize 50					running	running
21	AC	UGF	running	> 250M					running	running
- Parameters: vdd=1.8, M10.l=2u**

Point	Test	Output	Nominal	Spec	Weight	Pass/Fail	Min	Max	C0_0	C0_1
19	AC	/V1/PLUS								
19	AC	/OUT								
19	AC	gainBwProd(VF("/OUT"))	503.2M				466.8M	658.3M	658.3M	570.8M
19	AC	Current	389.7u	minimize 1m		pass	383u	415.8u	415.8u	401.4u
19	AC	Gain	46.18	maximize 50		near	45.73	47.59	47.59	46.85
19	AC	UGF	250.8M	> 250M		near	239.5M	311.5M	311.5M	279M
- Parameters: vdd=1.8, M10.l=1.9u**

Point	Test	Output	Nominal	Spec	Weight	Pass/Fail	Min	Max	C0_0	C0_1
18	AC	/V1/PLUS								

A status bar at the bottom indicates 'ManualTuning.0.Interactive.1'.

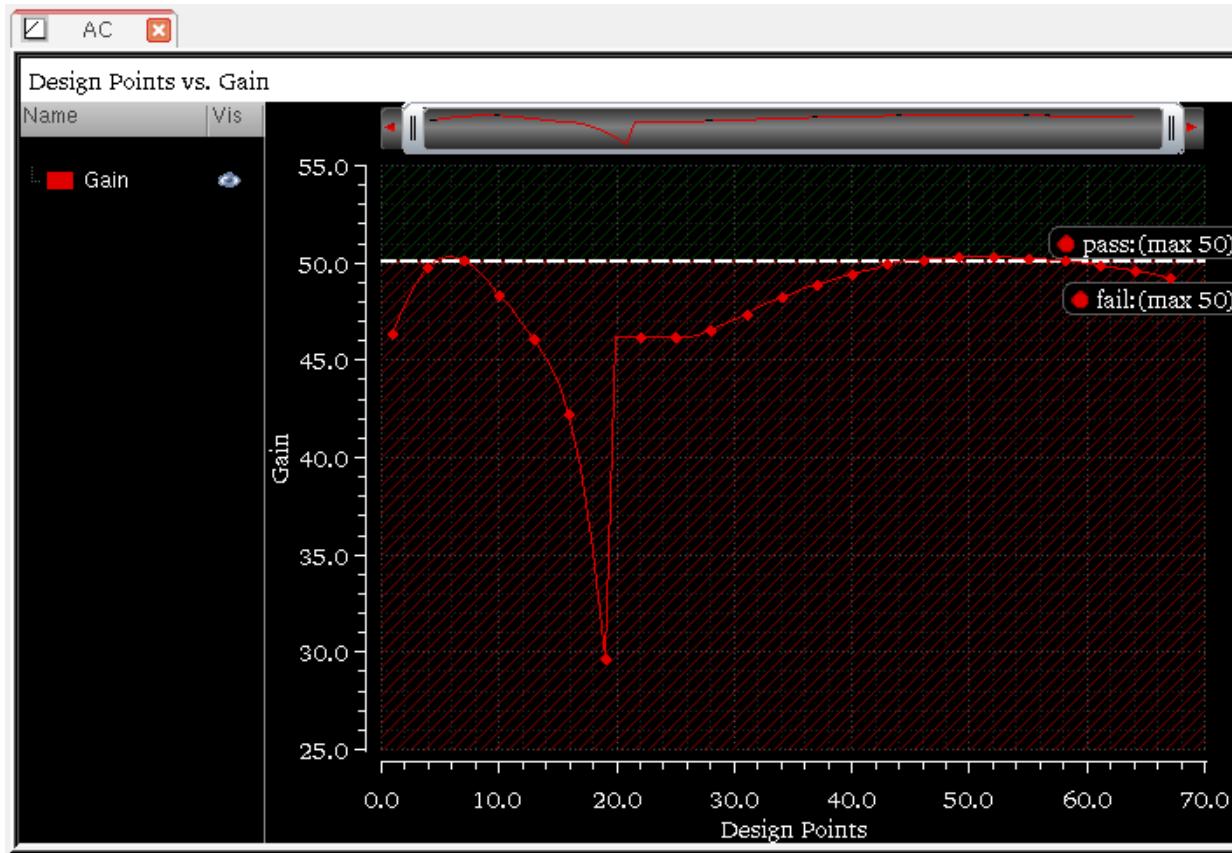
In the other run modes, results of every next simulation run are displayed on a new tab. Also, note that the name of the Results tab has also changed to ManualTuning.0.Interactive.1.

At the end of the second simulation run, the Results tab contains all the design points run during the Interactive.0 and Interactive.1 runs.

### 7. Compare the results of the two simulation runs.

You can also plot the results of all the simulations run in a manual tuning run. For that, in the Results tab, right-click on a specification and choose *Plot Across Design Points*.

Note that when you plot results across different design points, data is plotted for all the points of the two runs, as shown below.

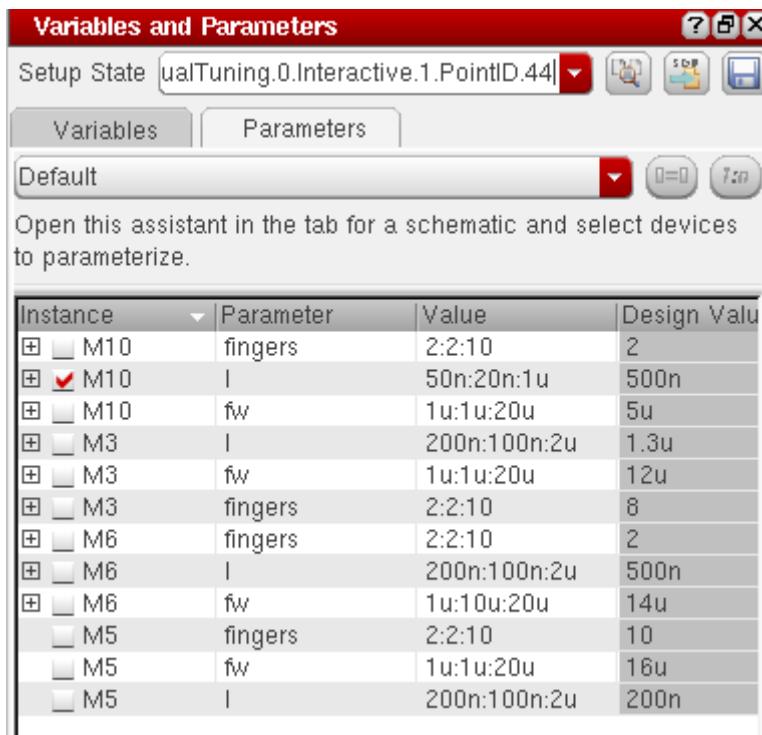


8. (Optional) If the desired results are not obtained, you might choose to further tune the design parameters and run more simulations till you get the best design point.
9. After the desired results are obtained, sort the design points to view the best or the worst point. If the best point meets the specifications, you can save the values used for that point so that they can be reused later. To do this, right-click on the gray row on top of that design point and choose *Save variable and parameter values to Setup State*.

Point	Test	Output	Nominal	Spec	Weight	Pass/Fail	Min	Max
Parameters: udd_1.8_M10_L530n								
44	AC	Backannotate						
44	AC	Save variable and parameter values to Setup State						
44	AC	Create Reference Point ...					1.089G	1.401G
44	AC	Submit Point ...				pass	860.6u	868.1u
44	AC	Troubleshoot Point				near	49.27	51.76
44	AC					pass	604.7M	679.9M

ADE XL creates a state by using the names of the simulation run and the design point ID. For example, for the design point 44 shown above, the tool creates a state named

ManualTuning.0.Interactive.1.PointID.44 and saves the values of variables and design parameters in that state.

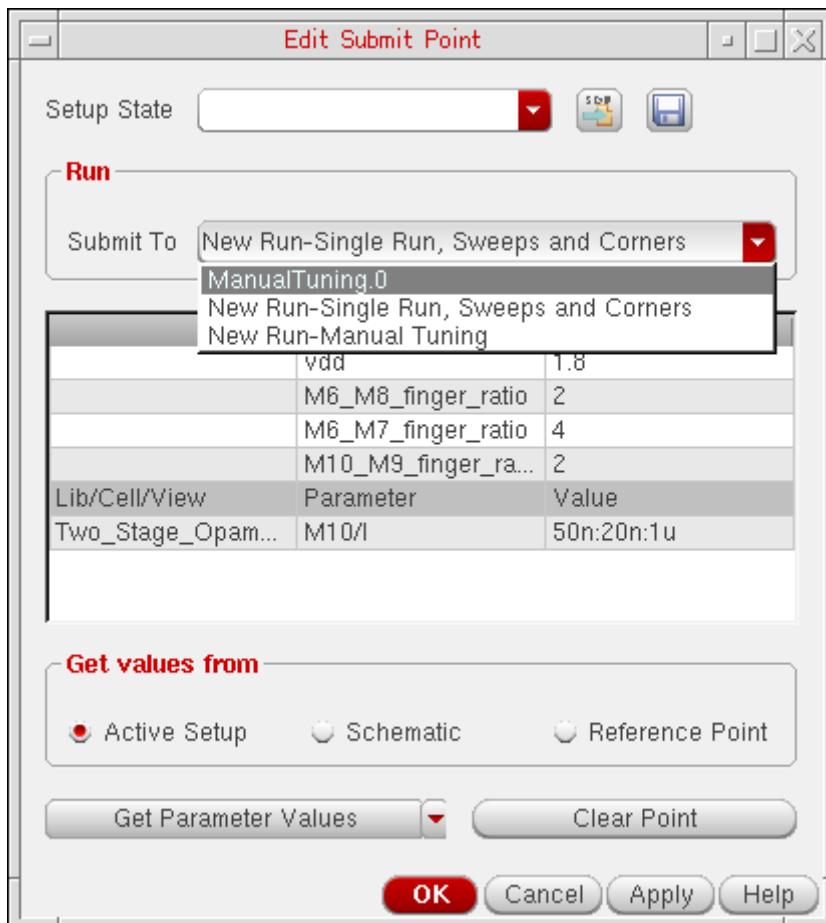


**Note:** You can use this saved state in the future to load the values of variables and parameters. For more details, see [Working with Global Variables in Analog Design Environment XL User Guide](#).

10. Click **Stop Simulation** on the *Run* toolbar to exit the Manual Tuning run mode.

**Important**

After exiting the Manual Tuning run mode, you can continue to run more simulations in a previous run by submitting more points to that run. To do this, you can open the **Edit Submit Point** form and submit a point to a previous manual tuning run listed in the *Submit To* drop-down list.



For example, if you choose `ManualTuning.0` in the form shown above and click **OK**, the `ManualTuning.0` run is started again, results are reloaded in the Results tab, and the new design points are appended to the existing results.

### **Important Notes**

- If you do not exit the Manual Tuning run mode and close the ADE XL GUI, the currently running Manual Tuning is not stopped. When you open a new ADE XL session, the Run Simulation button is yellow in color and you can continue with the same Manual Tuning run that was running earlier. If you want that the currently running Manual Tuning run should stop when the ADE XL GUI is closed, set the `stopManualTuningOnSessionExit` environment variable to `t`.
- Since the Manual Tuning results are saved to a single database, it is not possible to delete a specific child history or the simulation data for a specific child history. Therefore, on the Data View pane, the right-click menu commands, *Delete* and *Delete Simulation Data*, are not available for the child histories of a Manual Tuning run.

## Sizing Over Corners

While you can always size with all corners enabled using local or global optimization, it may not be efficient to do so with a large number of corners.

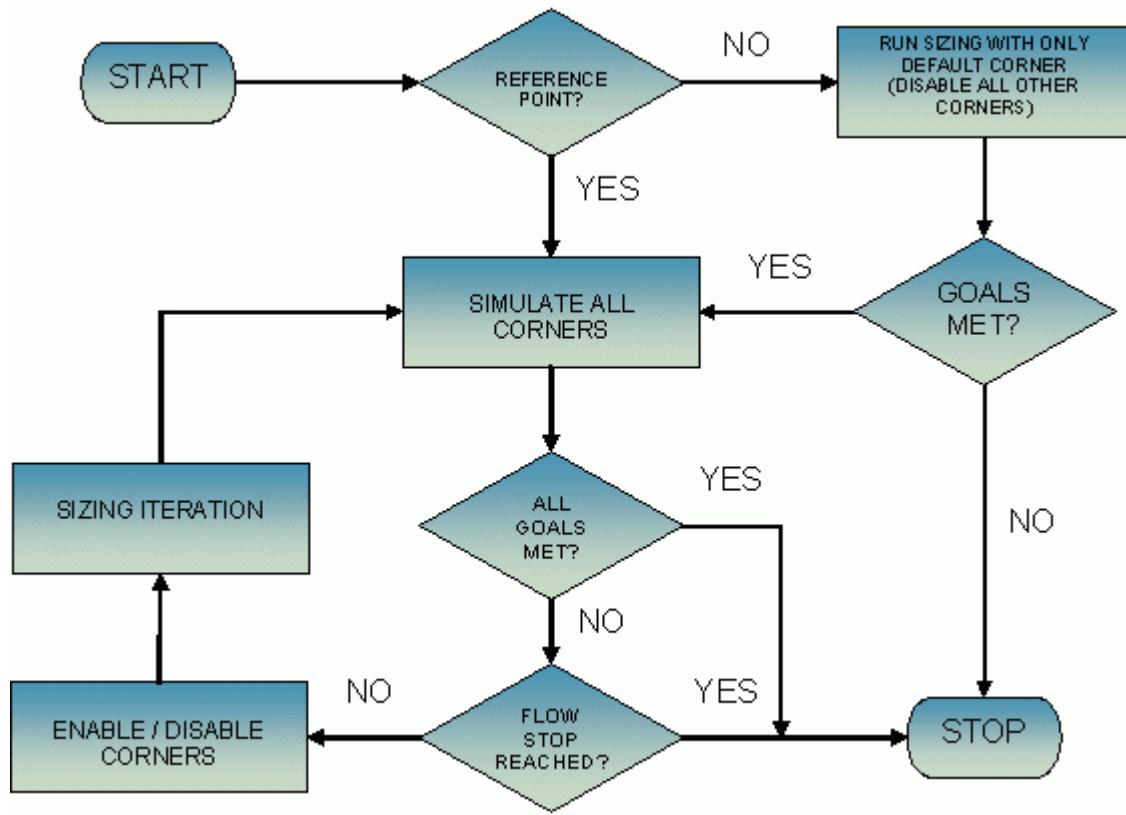
The Sizing Over Corners feature is an intelligent algorithm designed to optimize the test benches over a large number of corners. It does this by running a single point over all user-defined corners, determining the worst-case corners, using the worst-case corners for optimization and improving those corners on each iteration.

You can either provide a starting point, or ADE GXL performs an initial global optimization on the nominal corner and uses its best point as the starting point for the run. ADE GXL simulates all corners to determine the worst case performance corners for each goal. ADE GXL then resizes, taking into account the identified worst case performance corners. This process repeats until the stopping criteria for your size over corner run are met—either until all goals are met, or until it has run through the specified number of iterations or time limit.

**Note:** If the run stops due to the number of iterations or time limit being met, a final corner simulation is run if the last history children item for the run is related to sizing/optimization.

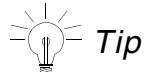
Figure 3-1 illustrates the sizing over corners flow.

**Figure 3-1 Flow for Sizing Over Corners**



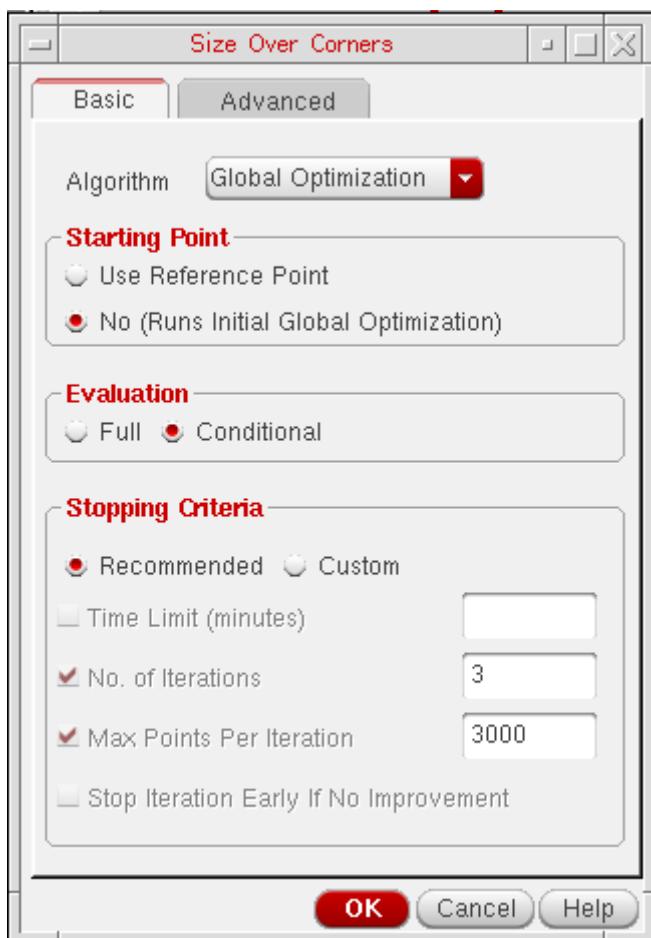
To size over corners, do the following:

1. From the *Run* menu, select *Size Over Corners*.



Alternatively, select *Size Over Corners* in the *Select a Run Mode* drop-down list on the *Run* toolbar, then click the *Simulation Options* button on the *Run* toolbar.

The Size Over Corners form appears.



2. From the *Algorithm* drop-down list on the Basic tab, select the algorithm for sizing over corners.

The default algorithm is *Global Optimization*.

3. If you have created a reference point, and want to use that point as the starting point for sizing, select *Use Reference Point* under *Starting Point*.

**Note:** You must have a reference point available to utilize this option.

Select *No (Runs Initial Global Optimization)* under *Starting Point* if you don't have a reference point, or do not want to use the reference point as the starting point for sizing. When this option is selected, ADE GXL performs an initial global optimization on the nominal corner and uses its best point as the starting point for the run. If you select *No (Runs Initial Global Optimization)*, ensure that:

- ❑ The nominal corner is not disabled in the Run Summary pane.

For more information, see the [Disabling and Enabling the Nominal Corner](#).

- For tests that are enabled in the Data View pane, the nominal corner is not disabled in the Corners Setup form.

For more information, see [Disabling and Enabling the Nominal Corner for Specific Tests](#).

4. In the *Evaluation* group box, select one of the following evaluation types for the sizing run:

- Full
- Conditional

For more information on conditional evaluation, see [Understanding Conditional Evaluation](#) on page 130.

5. Select the *Recommended* option under *Stopping Criteria* to use the recommended criteria for the length of time size over corners should run. The recommended criteria are:

- Three sizing iterations
- 3000 points run per iteration

If you want to modify these defaults, select the *Custom* option, then select one or more of the following stopping criteria:

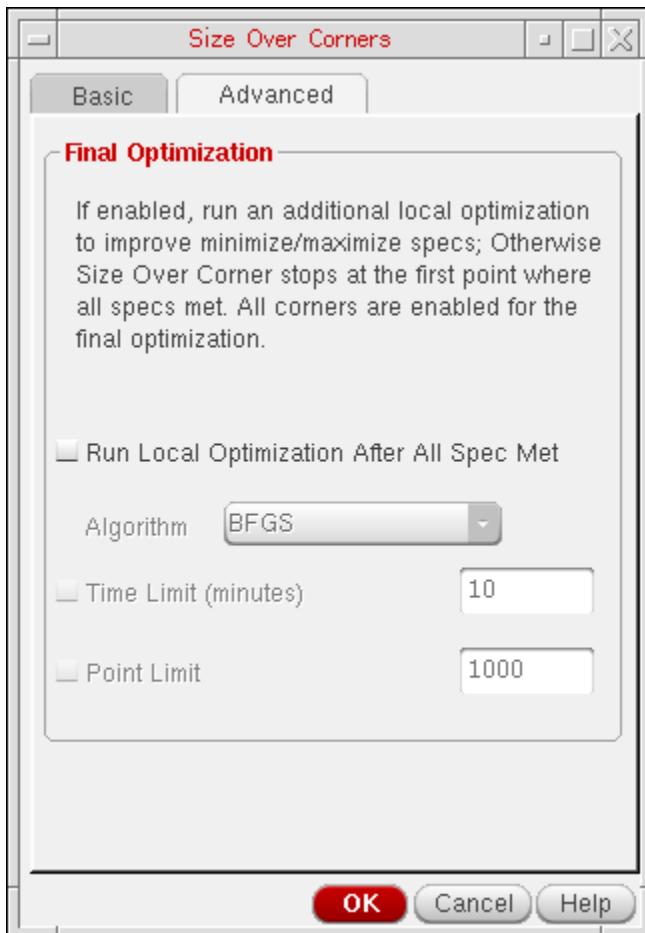
- To set a time limit for the run, select the *Time Limit (minutes)* check box and enter a value in minutes.
- To specify the number of sizing iterations, select the *No. of Iterations* check box and enter the number in the field.
- To specify the maximum number of points processed per iteration, select the *Max Points per Iteration* check box and enter the number of points in the field.
- To stop the process early if the sizing results in no improvement, select the *Stop Iteration Early if No Improvement* check box. This option is applied to each optimization iteration. This stopping criteria is similar to the *No Improvement with Points* stopping criteria for the global or local optimization run modes for which the user also specifies the number of points. In the case of iterative run modes, the number of points is calculated as *Max Points per Iteration* / 3.

**Note:** If all goals are met prior to the stopping criteria being reached, ADE GXL will stop the sizing run.

6. (Optional) By default, the Size Over Corners run is automatically stopped when all the goals are met. If the test outputs have minimize or maximize specs defined for them, you

can choose to automatically run the local optimization after the best design point is found. This helps in getting better results for these types of specifications.

For this, open the Advanced tab.



Select the *Run Local Optimization After All Spec Met* option. This specifies that local optimization needs to be run after all the specs are met. The other options on this tab are used by the local optimization run. If required, change the default values.

7. Click *OK* to save the changes and close the *Size Over Corners* form.
8. Click the *Run Simulation*  button on the Run toolbar.

ADE GXL runs through iterations of sizing and corner sweeping until your stopping criteria are met.

## Running Feasibility Analysis

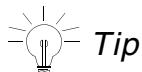
Feasibility analysis allows you to quickly verify that the operating region specifications can be satisfied. It does this by running optimization based on the operating region specifications set up for your circuit to evaluate if any design point passes the operating region specifications.

If feasibility analysis identifies design points that pass all operating region specifications, you can continue with tuning the design over other specifications or run optimization. For more information about running optimization, see [Running Optimization](#) on page 130. If no design point passes all operating region specifications, it does not mean that there are absolutely no feasible points in the solution space. In this case, you may change the design constraints or the circuit topology and then check if any design point passes all operating region specifications.

**Note:** Feasibility analysis evaluates only the operating region specifications. The tests, analyses and specifications that are not related to operating region specifications are not evaluated when you run feasibility analysis.

To run feasibility analysis, do the following:

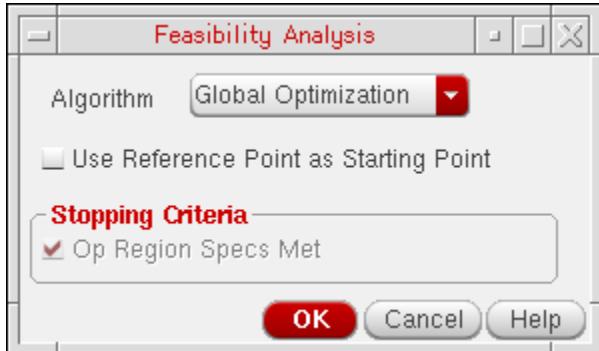
1. Specify operating region specifications for your circuit. For more information, see [Running Optimization](#) on page 130.
2. From the *Run* menu, select *Feasibility Analysis*.



*Tip*

Alternatively, select *Feasibility Analysis* in the *Select a Run Mode* drop-down list on the Run toolbar, then click the *Simulation Options* button on the Run toolbar.

The Feasibility Analysis form appears.



3. From the *Algorithm* drop-down list, select the algorithm for optimizing the design to meet the operating region specifications.

The following algorithms are supported:

- Global Optimization
- Conjugate Gradient
- Brent-Powell
- Hooke-Jeeves

4. Select the *Use Reference Point as Starting Point* check box if you have created a reference point and want to use that point as the starting point for the run.

Note the following:

- You must have a reference point when the specified algorithm is *Conjugate Gradient, Brent-Powell* or *Hooke-Jeeves*.
- Selecting the *Use Reference Point as Starting Point* check box is optional when the specified algorithm is Global Optimization.

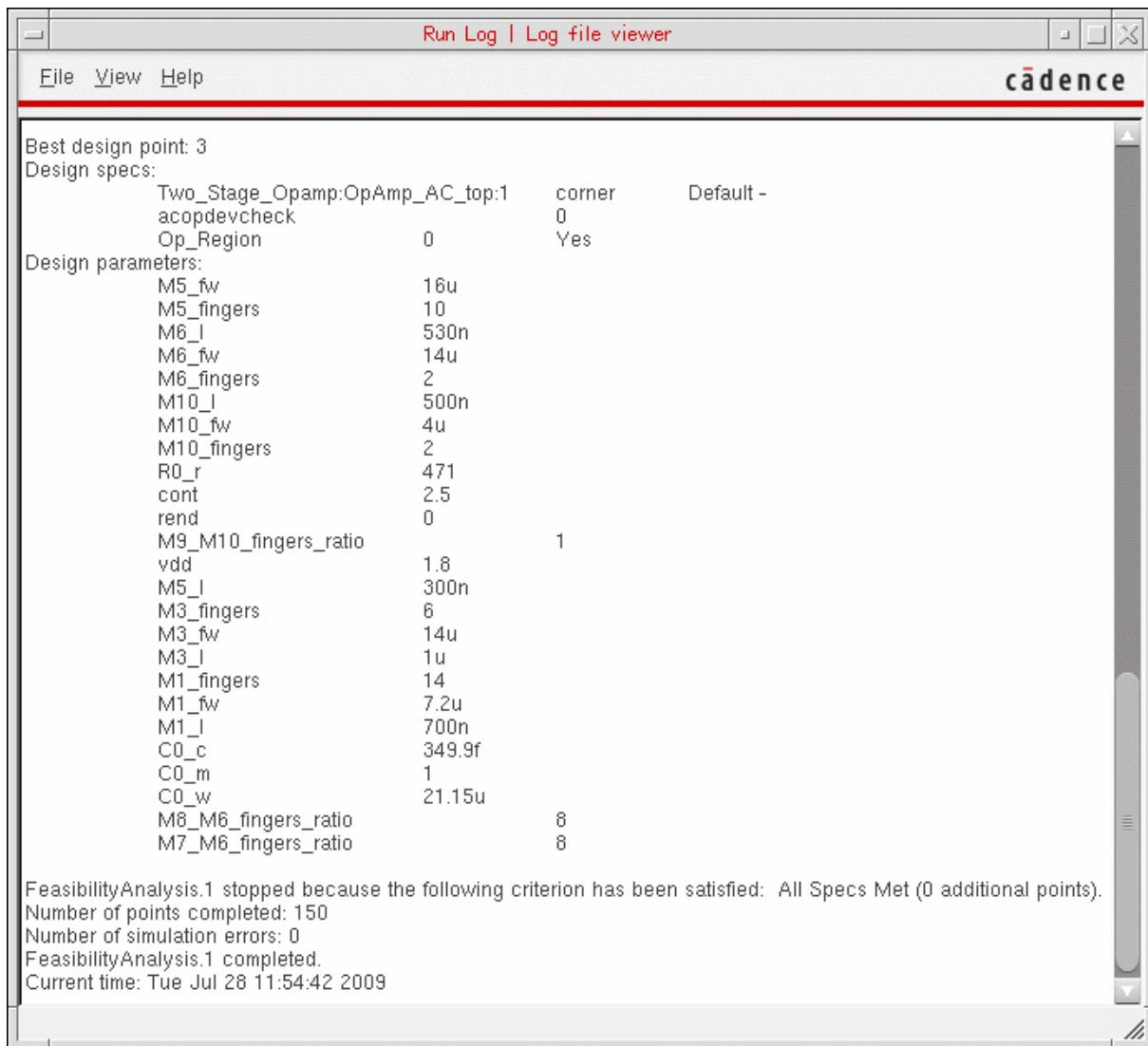
5. Click *OK* to close the Feasibility Analysis form and start the feasibility analysis run.

**Note:** If you opened the Feasibility Analysis form by clicking the *Simulation Options*  button on the Run toolbar, click the *Run Simulation*  button on the Run toolbar to start the feasibility analysis run.

# Virtuoso Analog Design Environment GXL User Guide

## Circuit Optimization

Information regarding design points that pass all operating region specifications are displayed in the Run Log | Log File Viewer form.



### Important

A feasibility analysis run stops only when all operating region specifications are met. If you want to stop the run before this is achieved, click the *Stop Simulation*  button on the *Run* toolbar.

## Improving the Yield

The *Improve Yield* command can be used to return a design point that meets all corners and has the highest possible yield.

This command runs iterations of sizing and Monte Carlo analysis to arrive at a solution. When you start Improve Yield, ADE GXL first generates the statistical corners, then, as the run progresses, evaluates points on a subset of those corners. Promising points are then evaluated on a larger set of corners. Eventually ADE GXL arrives at the best point -- one that has been evaluated at all statistical corners and has the highest possible yield.

Also available are a number of stopping criteria, including time and points limits. Once ADE GXL hits any of the specified options, it will end the improvement process.



*Improve Yield*, like *Monte Carlo Sampling*, is available only for the Spectre circuit simulator.

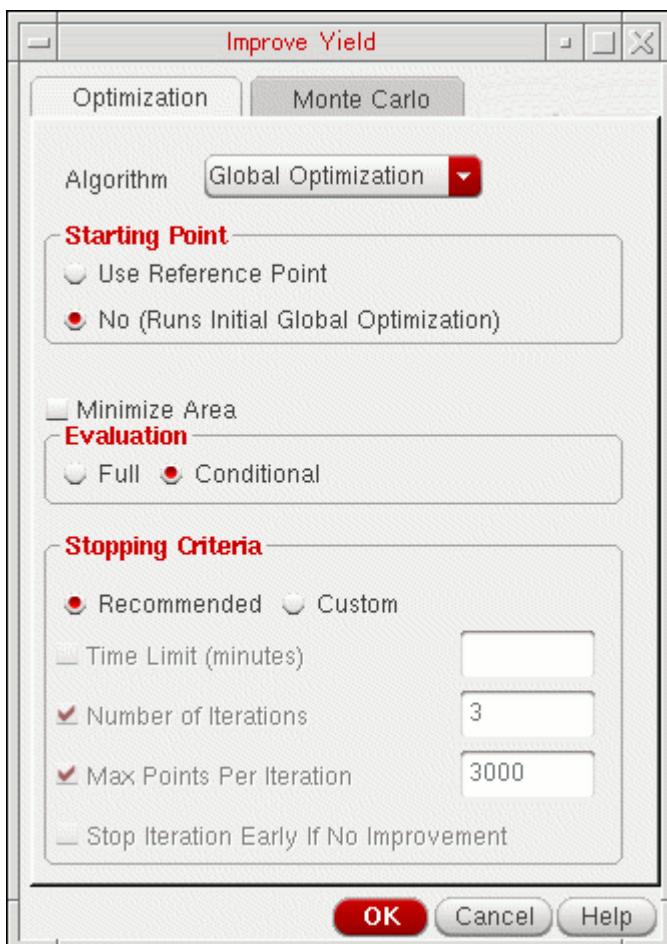
Note that your design must include devices or device models for which you have specified statistically varying parameter values. You must have one or more specs defined and enabled. You must specify either global (process) or mismatch (per-instance) variations or both. You can also specify correlation information. After simulating, you can select the yield view to view mean and standard deviation information. For more information, see “Performing Monte Carlo Analysis” in the *Virtuoso Analog Design Environment XL User Guide*.

**Note:** For information about specifying parameter distributions for Spectre circuit simulation, see Specifying Parameter Distributions Using Statistics Blocks in the Analyses chapter of the *Spectre Circuit Simulator User Guide*.

To improve the yield:

1. From the *Select a Run Mode* drop-down list on the Run toolbar, choose *Improve Yield*.
2. Click to specify improve yield options.

The Improve Yield options form appears.



3. On the Optimization tab, select the optimization algorithm from the *Algorithm* drop-down list.
4. (Optional) If you have created a schematic point or [a reference point](#), and you want to use that point as a starting place for sizing, select the *Use Reference Point* option.  
You must have [a reference point](#) available to utilize this option.
5. Select an evaluation type by selecting one of the following radio buttons under *Evaluation*:
  - Full
  - Conditional

For more information on conditional evaluation, see [Understanding Conditional Evaluation](#).

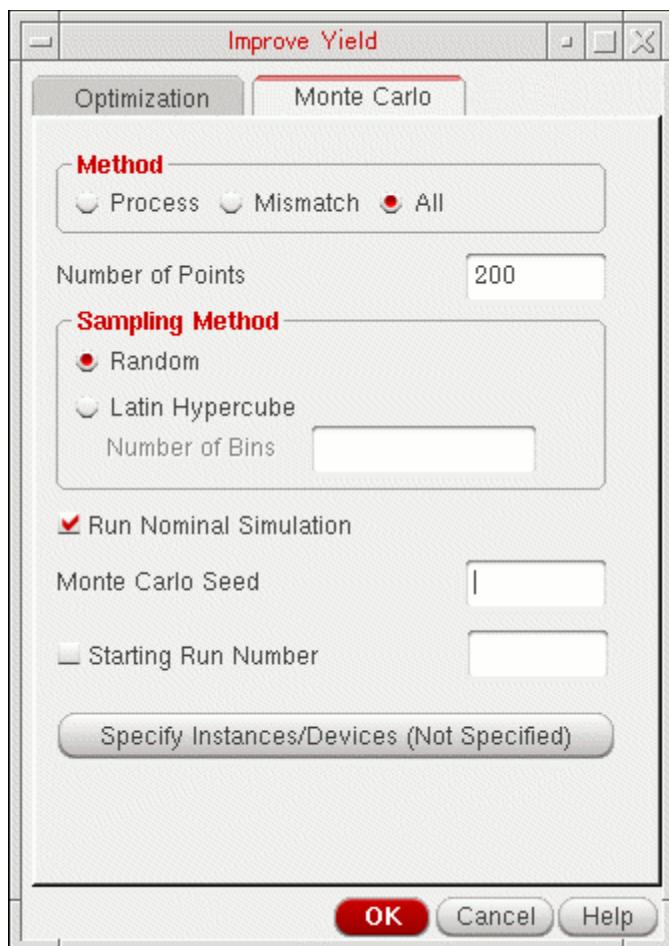
6. Select the *Recommended* check box under *Stopping Criteria* to use the recommended options. The recommended options are:

- Three sizing/Monte Carlo iterations
- 3000 points run per iteration

If you want to modify these defaults, select the *Custom* check box, then one or more of the following stopping criteria:

- a. To set a time limit for the run, select the *Time Limit (minutes)* check box and enter a value in hours.
- b. To specify the number of sizing/Monte Carlo iterations, select the *No. of Iterations* check box and enter the number in the field.
- c. If you want to specify the maximum number of points processed per iteration, select the *Max Points per Iteration* check box and enter the number of points in the field.
- d. If you want to stop the process early if the sizing results in no improvement, select the *Stop Iteration Early if No Improvement* check box. This option is applied to each optimization iteration. This stopping criteria is similar to the *No Improvement with Points* stopping criteria for the global or local optimization run modes for which the user also specifies the number of points. In the case of iterative run modes, the number of points is calculated as *Max Points per Iteration* / 3

7. Select the Monte Carlo tab to specify options for Monte Carlo.



8. When you run Improve Yield, you have a choice of varying the process statistical variables, mismatch statistical variables, or both. If you run only one type of statistical variable, the other variables are set to fixed values.

In the *Method* group box, select one of the following statistical variations:

*Process* for process statistical variations

*Mismatch* for per-instance statistical variations

*All* for both process and per-instance statistical variations



*Important*

You must define your models so that they respond to the statistical variations you choose. You must specify the file containing your models on the [Model Library Setup form](#). For a Spectre circuit simulator example of how to define your models, see “Specifying Parameter Distributions Using Statistics Blocks” in the *Virtuoso Spectre Circuit Simulator User Guide*.

9. In the *Number of Points* field, type the number of Monte Carlo points you want to simulate.
10. In the *Sampling Method* group box, select the statistical sampling method to be used—*Random* or *Latin Hypercube*.
11. If the selected sampling method is *Latin Hypercube*, specify the number of bins (subdivisions) for the *Latin Hypercube* method in the *Number of Bins* field.

Note the following:

- If a number is specified, the number of bins will be the specified number, or *Number of Points* + *Starting Run Number* - 1, whichever is greater. For example, if the specified number of bins is 90, the number of points specified in the *Number of Points* field is 100 and the starting run number specified in the *Starting Run Number* field is 6, the value 105 (100+6-1) is used.
  - If no number is specified, a default value of *Number of Points* + *Starting Run Number* - 1 is used. For example, if the number of points specified in the *Number of Points* field is 100 and the starting run number specified in the *Starting Run Number* field is 6, the default value of 105 (100+6-1) is used.
12. If you want to run a simulation at the reference point prior to beginning the improve yield process, select the *Run Nominal Simulation* check box.

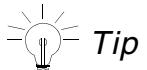
When this option is selected, Spectre will run a simulation at the reference point, and, if this fails, then the sampling process is not initiated and the simulation stops.

13. (Optional) If you want to specify a different seed for the Monte Carlo analysis, select the *Monte Carlo Seed* check box and enter the seed number.

By always specifying the same seed, you can reproduce a previous experiment. If you do not specify a seed, the value 12345 is used.

14. (Optional) If you want to specify a starting run number, select the *Starting Run Number* check box and enter the starting run number.

The starting run number specifies the run that Monte Carlo begins with. By specifying this number, you can reproduce a particular run or sequence of runs from a previous experiment (for example, to examine an earlier case in more detail).



*Tip*

To reproduce a run or sequence of runs, you need to specify the same value in the Starting Run # and the Monte Carlo Seed fields.

15. By default, mismatch variations are applied to all subcircuit instances in the design. Click the *Specify Instances/Devices* button to specify the sensitive instances and devices you want to either include or exclude for applying mismatch variations. For more information, see the “[Including or Excluding Instances and Devices for Applying Mismatch Variations](#)” section in the [Virtuoso Analog Design Environment XL User Guide](#).

**Note:** Starting from IC6.1.5 ISR15, the N-Sigma tab is not available on the Improve Yield options form. You can now run High Yield Estimation and create statistical corners to optimize your design and to achieve a yield of less than the desired sigma value for the selected specifications. For more details, refer to [Creating Statistical Corners From a Worst Case Distance Analysis](#).

16. In the Run workspace, click the *Run simulation* icon to improve the yield.

ADE GXL begins synthesizing with Monte Carlo analysis.

When the Improve Yield run is finished, the Data View lists the Improve Yield check point. Expanding this check point displays the different runs that make up a full Improve Yield run, including iterations of Optimization and Monte Carlo. You can view the results of any of these runs by right-clicking and choosing *View Results*.

## **Creating, Viewing, and Modifying Reference Points**

You can create a reference point for [Improve Yield](#), [Global Optimization](#), [Monte Carlo Sampling](#), or [Sensitivity Analysis](#) from scratch or based on a point in a run. You can also view and modify the [current reference point](#) or a [reference point from a run](#).

For more information, see the following topics:

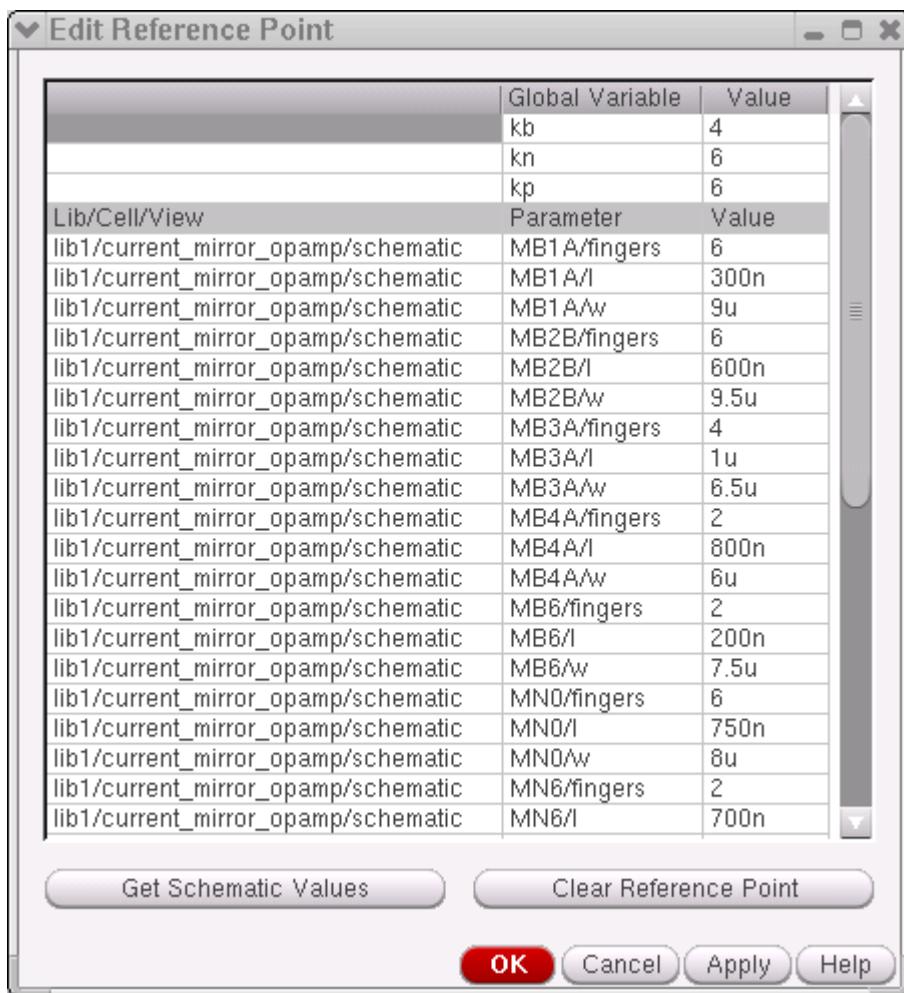
- [Creating a Reference Point from Scratch](#) on page 157
- [Creating, Viewing, and Modifying a Reference Point from a Run](#) on page 159
- [Viewing and Modifying the Current Reference Point](#) on page 161

### **Creating a Reference Point from Scratch**

To create a reference point from scratch:

1. Click the *Edit Reference Point*  icon in the Run Mode tool bar, or choose *Edit Reference Point* from the *Run* menu.

The Edit Reference Point form appears. If you have not previously created a reference point or made any optimization runs, this form will be populated with the schematic values.



**2. Add and modify the values as desired:**

- Select the Value field for any parameter and enter a value.
- To pull in the schematic values do the following:
  - To pull in the schematic values for specific parameters, select the parameters for which you want the schematic values, click *Get Schematic Values* and choose *Selected Parameters*.
  - To pull in the schematic values for all the parameters that have no values, click *Get Schematic Values* and choose *Parameters With No Value*.

- To pull in the schematic values for all parameters, click *Get Schematic Values* and choose *All Parameters*.
  - Select one or more rows and click *Clear Reference Point* to clear the values.
3. Click *OK* to save your values and dismiss the form.

Any reference point you create will remain the active reference point until you create or modify another point.

You can now utilize this point by selecting the *Use Reference Point as Starting Point* check box for Improve Yield or Global Optimization.

## Creating, Viewing, and Modifying a Reference Point from a Run

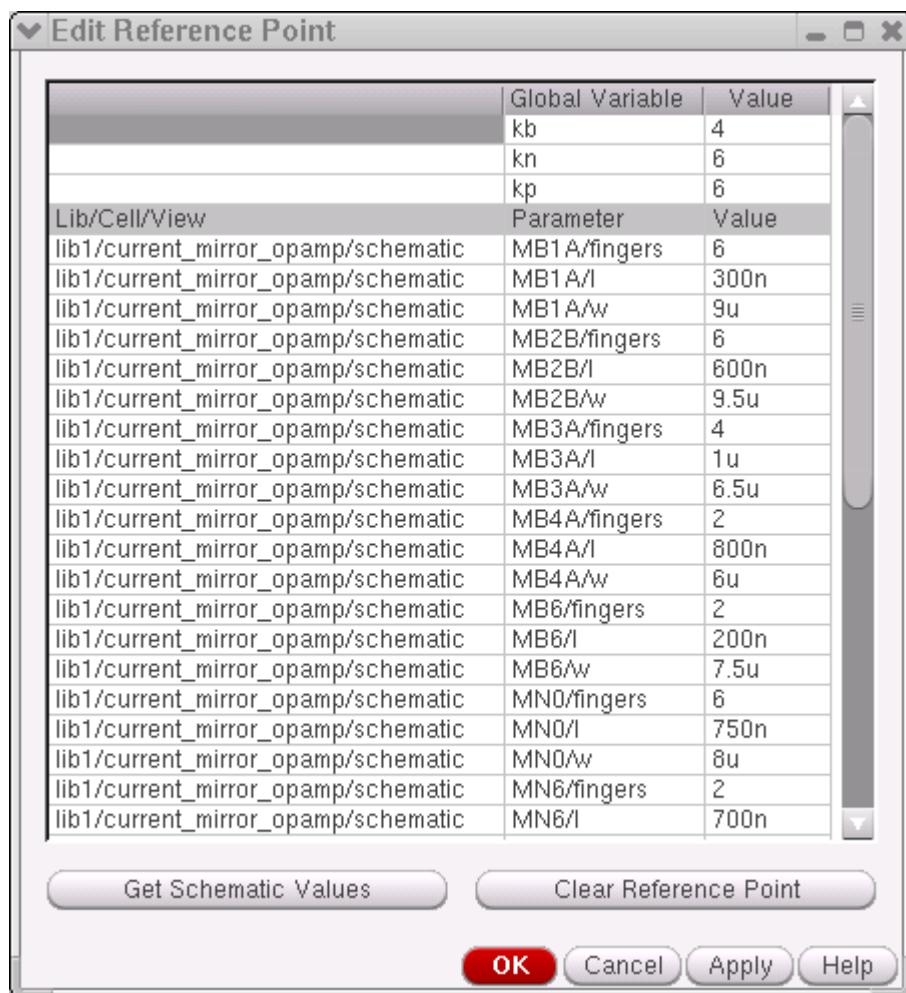
To create, view, or modify a reference point from a run:

1. Right-click on a point (highlighted in gray) in the Results tab and choose *Create Reference Point*.

The screenshot shows the 'Results' tab of the Virtuoso Analog Design Environment. At the top, there are tabs for 'Outputs Setup', 'Results', and 'Diagnostics'. Below the tabs is a toolbar with buttons for 'View: Optimization', 'Show', 'Evaluate', 'Plot All', 'Replace', and 'Export'. The main area is a table with columns: Point, Test, Output, Value, Spec, Weight, Min, and Max. A row in the table is highlighted in green, indicating it is selected. A callout bubble labeled 'Reference Point' points to this green-highlighted row. The table also contains a header row and several data rows. The 'Value' column for the highlighted row shows '63.32'.

Point	Test	Output	Value	Spec	Weight	Min	Max
Parameters: kb=4, kn=6, kp=4, MB1A.fingers=6, MB1A.I=300n, MB1A.w=9u, MB2B.fingers=6, MB2B.I=300n, MB2B.w=9u							
235	lib1:OpAmp_AC_top:1	phasemargin	23.99	info	22.52	22.52	25.00
235	lib1:OpAmp_AC_top:1	DCgain	63.32	> 60	62.41	62.41	65.00

The Edit Reference Point form appears.



2. If desired, modify the values as described in [step 2 of Creating, Viewing, and Modifying Reference Points](#):
3. Click **OK** to save your values and dismiss the form.

If you are merely viewing the point, and haven't made any changes you want to save, click **Cancel**.

Any reference point you create will remain the active reference point until you create or modify another point.

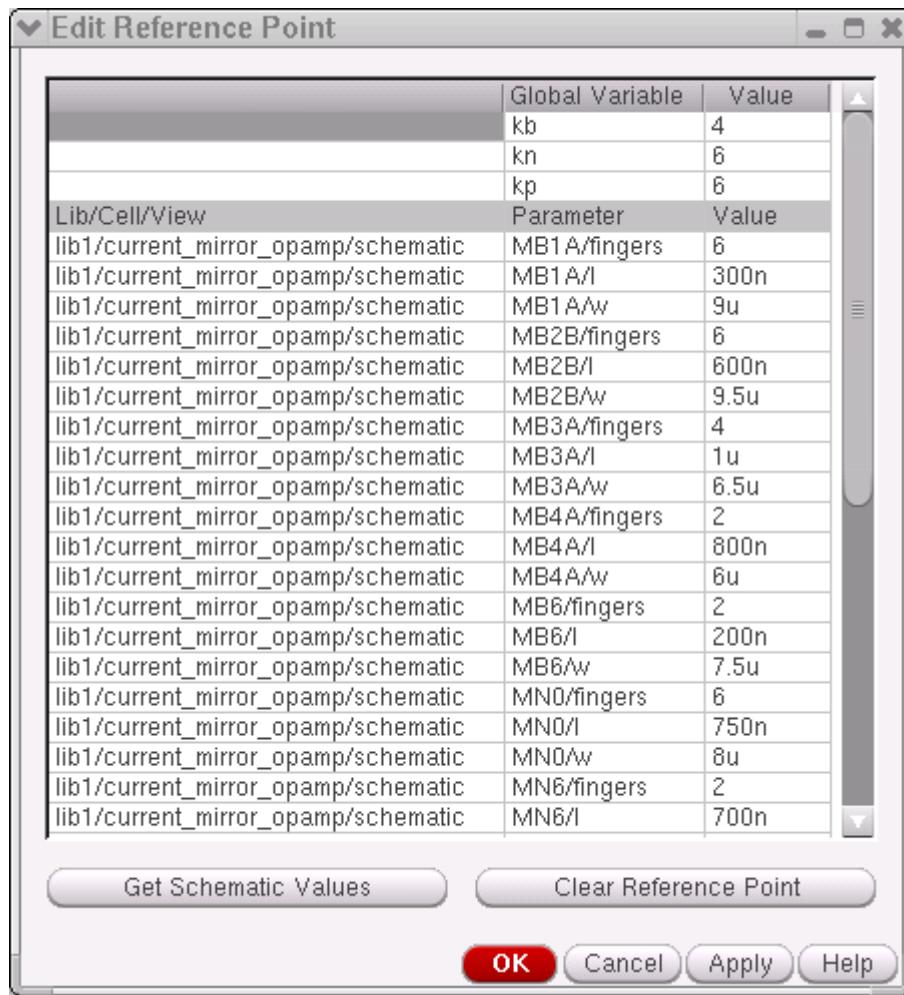
You can now utilize this point by selecting the *Use Reference Point as Starting Point* check box for [Improve Yield](#) or [Global Optimization](#).

## **Viewing and Modifying the Current Reference Point**

The current reference point is the last one created or modified, then saved to the setup database. You can view and modify this current reference point.

1. Click the *Edit Reference Point* icon in the Run Mode tool bar.

The Edit Reference Point form appears, displaying the values for the current reference point.



### **Important**

If redundant parameters exist, they are shaded in yellow, and you will be prompted to delete them. Redundant parameters are parameters that are present in the starting point but are either deleted or disabled in the design space.

2. If desired, modify the values as described in step 2 of Creating, Viewing, and Modifying Reference Points:
3. Click *OK* to save your values and dismiss the form.

If you are merely viewing the point, and haven't made any changes you want to save, click *Cancel*.

Any reference point you create will remain the active reference point until you create or modify another point.

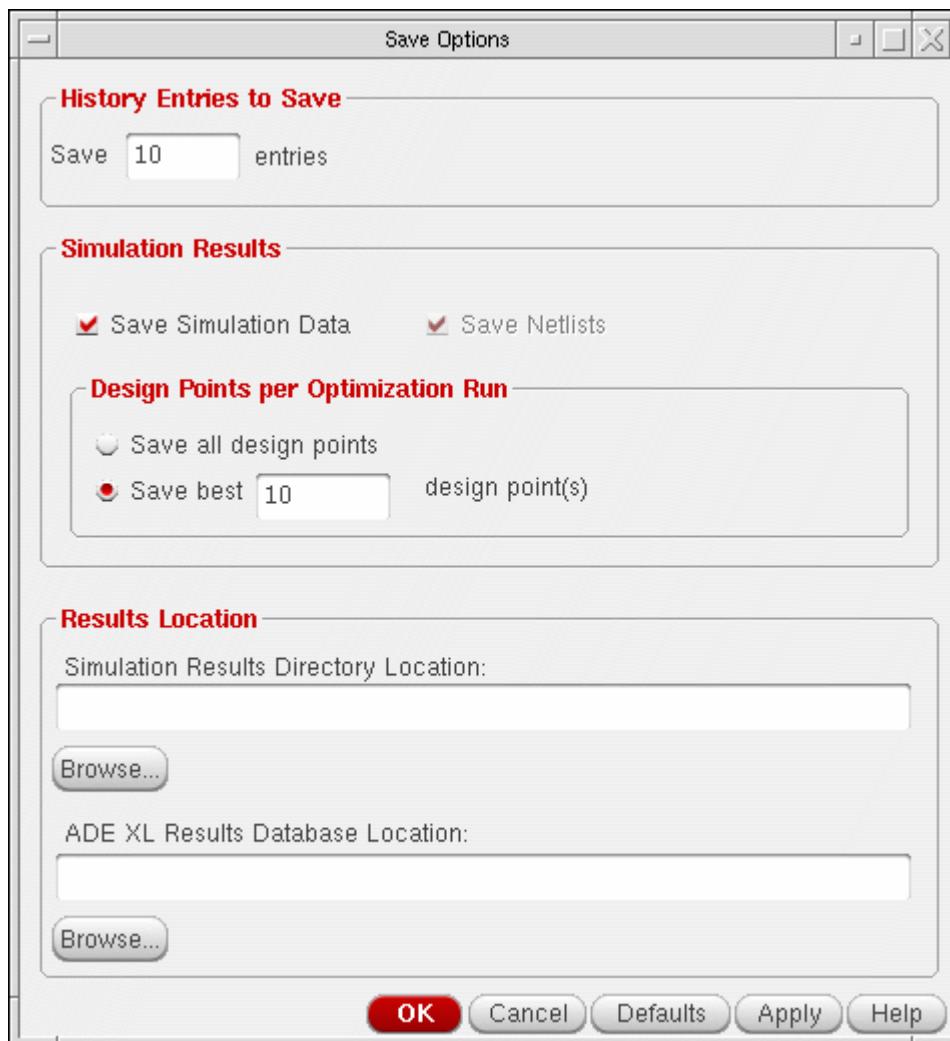
You can now utilize this point by selecting the *Use Reference Point as Starting Point* check box for Improve Yield or Global Optimization.

## Specifying How Much Optimization Data to Save

To specify the number of design points for which you want to save simulation data, do the following:

1. In the ADE GXL session window, choose *ADE GXL – Options – Save*.

The Save Options form appears.



2. In the *Design Points per Optimization Run* group box, select one of the following radio buttons:

<b>Radio Button</b>	<b>Description</b>
<i>Save all design points</i>	Saves data from all <u>design points</u>
<i>Save best</i>	Saves data from the specified number of best <u>design points</u>

3. (Optional) In the *Results DB* group box, either type the directory path where you want the program to write your setup and results database information in the *Results DB location* field, or do the following:

- a. Click *Browse*.
- b. On the form that appears, navigate to and select the directory where you want the program to write your setup and results database information.

The program writes setup and results database information to *libraryName/cellName/adexl* in the specified directory.

If you do not specify a setup and results database location, the program writes this information to *libraryName/cellName/adexl* for the current test. If your design library is set up as read-only, you can use this field to specify a writable location.

4. Click *OK*.

The program applies the settings you specified.

See also

- [Data Points—Definition](#) on page 165
- [Design Points—Definition](#) on page 165
- [Specifying Results Database Location](#) in the *Virtuoso Analog Design Environment XG User Guide*.

## Data Points—Definition

A data point represents one simulation run with one set of parameter values and corners setup. For example, if you sweep *CAP* (a global variable) from 600p to 800p with a step value of 3, you will have three data points. If you sweep more than one variable, each unique combination of values constitutes one data point. If you have two corners for temperature at 0 and 30 in addition to the *CAP* sweep through three values, you will have six data points:

Data Point	<i>CAP</i>	Corner Temperature
1	600p	0
2	700p	0
3	800p	0
4	600p	30
5	700p	30
6	800p	30

## Design Points—Definition

A design point consists of the set of data points that represents one sweep value across all corners. For example, if you have two corners for temperature at 0 and 30, and a sweep of *CAP* (a global variable) through three values (600p, 700p, 800p), each sweep across both corners constitutes a design point.

Design Point	<i>CAP</i>	Corner Temperatures
1	600p	0
	600p	30
2	700p	0
	700p	30
3	800p	0
	800p	30

# **Virtuoso Analog Design Environment GXL User Guide**

## Circuit Optimization

---

---

## High Yield Estimation

---

High Yield Estimation is a supplement to the Monte Carlo Sampling run mode. It is recommended for estimating the yield of high sigma designs, where the simulation cost of Monte Carlo analysis is high. This chapter describes how to perform high yield estimation by using ADE GXL.

This chapter covers the following topics:

- [Performing High Yield Estimation](#) on page 168
- [Creating Statistical Corners From a Worst Case Distance Analysis](#) on page 179

## Performing High Yield Estimation

High Yield Estimation finds the shortest distance—referred to as Worst Case Distance or WCD—from the nominal point to the specification boundary in the process/mismatch parameter space. The worst case distance is a good indicator of circuit yield, where yield in percentage is approximately equal to

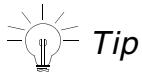
$$\frac{1}{\sqrt{2}\pi} \int_{-\infty}^{wcd} e^{-t^2/2} dt = \frac{1}{2} \left[ 1 + \operatorname{erf} \left( \frac{wcd}{\sqrt{2}} \right) \right]$$

where,  $\operatorname{erf}$  is the error function.

**Note:** WCD provides accurate yield estimation when the specification boundary is linear in the process or mismatch parameter space. Strong non-linearity of the specification boundary can cause difficulty in finding the WCD points. A non-linear specification however may not result in a non-linear specification boundary in statistical space.

High Yield Estimation supports only statistical parameters that follow a normal distribution. It begins with a Monte Carlo Sampling run, uses the Monte Carlo results to filter non-high yield specifications (for which the Monte Carlo run gives accurate yield estimates), and then applies the WCD method on each high yield specification by doing the following:

1. Reads the process and mismatch parameter information from the Monte Carlo results
2. Performs parameter reduction based on the Monte Carlo results
3. Runs multiple sensitivity analysis iterations to find the WCD



You can improve the circuit yield by creating a statistical corner from the worst case distance point. For more details, refer to [Creating Statistical Corners From a Worst Case Distance Analysis](#) on page 179.

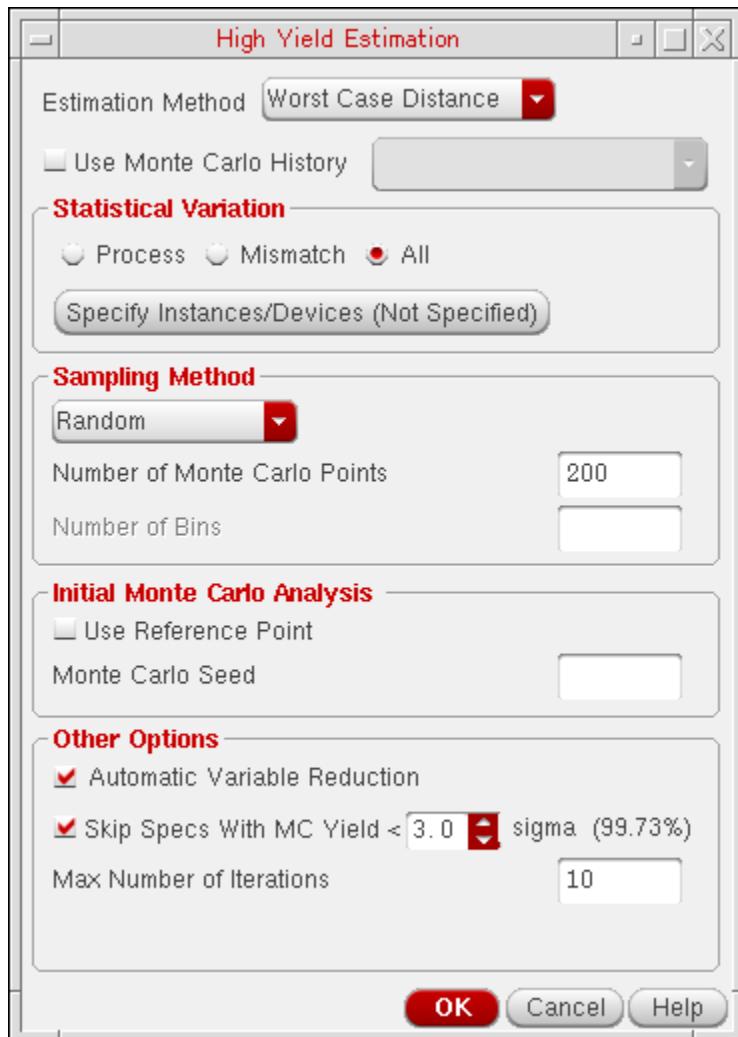
To perform High Yield Estimation, do the following:

1. Ensure that range and tolerance (`to1`) type specifications are disabled or deleted in the Outputs Setup tab. This is because these two specifications are not supported for High Yield Estimation. You can also convert the range and tolerance (`to1`) type specifications into two separate specifications for the min and max boundaries. For more information about specifications, see [Specifying General Specifications](#).
2. Create a [reference point](#) if swept values are specified for enabled global variables or parameters.

**3. Choose Run – High Yield Estimation.**

Alternatively, select *High Yield Estimation* from the *Select a Run Mode* drop-down list on the Run toolbar, and then click the *Simulation Options*  button on the Run toolbar.

The High Yield Estimation form appears.



4. The *Estimation Method* field is by default set to *Worst Case Distance*. Currently, this is the only option used by High Yield Estimation.
5. If you have already run a Monte Carlo simulation, you can use the process and mismatch data from the history of that run. For that, select the *Use Monte Carlo History* option and select a reference Monte Carlo run history from the list of available histories given in the drop-down list next to this option.

It is essential that the simulation data of the selected history contains the process and mismatch data. If any one of these data is not available, the following error message is displayed.



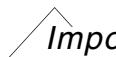
In this case, you can clear the *Use Monte Carlo History* option and run a fresh Monte Carlo simulation or choose another history that contains the process and mismatch data.

**Note:** When you choose to use an existing Monte Carlo history as reference, you can only specify options in the *Other Options* section. Values for the remaining options are taken from the referenced Monte Carlo history.

6. When you run High Yield Estimation, you have a choice of varying the process statistical variables, mismatch statistical variables, or both. If you run only one type of statistical variable, the other variables are set to fixed values.

In the *Method* group box, select one of the following statistical variations:

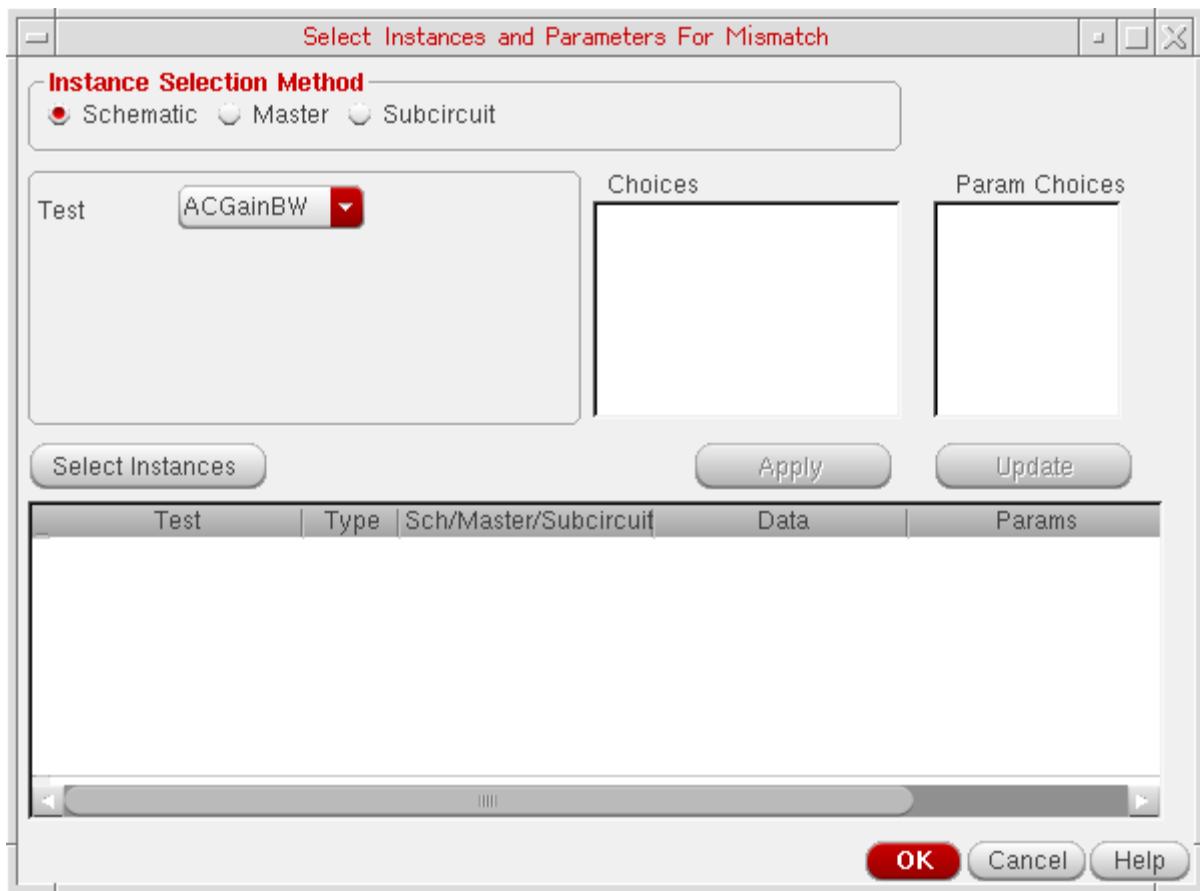
- |                 |  |
|-----------------|--|
| <i>Process</i>  | for process statistical variations                       |
| <i>Mismatch</i> | for per-instance statistical variations                  |
| <i>All</i>      | for both process and per-instance statistical variations |

 **Important**

You must define your models so that they respond to the statistical variations you choose. You must specify the file containing your models on the [Model Library Setup](#) form. For a Spectre circuit simulator example of how to define your models, see “Specifying Parameter Distributions Using Statistics Blocks” in *Virtuoso Spectre Circuit Simulator User Guide*.

7. By default, mismatch variations are applied to all subcircuit instances in the design. Click the *Specify Instances/Devices* button to specify the sensitive instances, devices, and parameters that you want to either include or exclude for applying mismatch variations.

The Select Instances and Parameters for Mismatch form is displayed, as shown in the following figure.



For more information on how to use this form, refer to [Selecting Instances and Parameters for Mismatch Variation](#).

8. From the *Sampling Method* list, select a statistical sampling method from the drop-down list. You can select any one of the following sampling methods:
  - Random*: In this method, samples are randomly selected without considering the other selected samples.
  - Latin Hypercube*: In this method, for each variable, the sampled distribution is divided into bins of equal probability. A random value within each bin is then selected for each variable. The samples from each bin are randomly matched to create multi-dimensional sample points. This method ensures that the ensemble of sample points is representative of the real variability.
9. If the selected sampling method is *Latin Hypercube*, specify the number of bins (subdivisions) for the *Latin Hypercube* method in the *Number of Bins* field.

Note the following:

- If a number is specified, the number of bins will be the specified number, or *Number of Points + Starting Run Number - 1*, whichever is greater. For example, if the specified number of bins is 90, the number of points specified in the *Number of Points* field is 100 and the starting run number specified in the *Starting Run Number* field is 6, the value 105 (100+6-1) is used.
  - If no number is specified, a default value of *Number of Points + Starting Run Number - 1* is used. For example, if the number of points specified in the *Number of Points* field is 100 and the starting run number specified in the *Starting Run Number* field is 6, the default value of 105 (100+6-1) is used.
10. In the *Number of Monte Carlo Points* field, type the number of Monte Carlo points you want to simulate.
11. Select the *Use Reference Point* check box if you have created a reference point in step 2 and want to use that point as the starting point for the run.
12. (Optional) If you want to specify a different seed for the Monte Carlo analysis, enter the seed number in the *Monte Carlo Seed* field.  
By always specifying the same seed, you can reproduce a previous experiment. If you do not specify a seed, the value 12345 is used.
13. By default, the *Automatic Variable Reduction* check box is selected. This option reduces the set of statistical variables by eliminating insignificant variables (variables that have no variation or have no influence on the WCD point).
  - Clear this selection if you do not want to reduce the statistical variables. Insignificant variables bring noise and require more simulations for sensitivity analysis. Therefore, it is recommended to enable variable reduction.
14. To ignore the specifications for which Monte Carlo yield is less than a specified percentage, select the *Skip the Specs Whose MC Yield <* checkbox and specify a sigma value.  
The default value of this field is 3 sigma; therefore, specifications for which the Monte Carlo yield is less than 99.73% are ignored.
15. If you want to run high yield estimation on all the specifications, disable the *Skip the Specs Whose MC Yield <* check box.
16. In the *Max Number of Iterations* field, specify the maximum number of iterations to be run for each specification.  
The default number of iterations is 10.
17. Click *OK* to close the High Yield Estimation form and start the high yield estimation run.

## Virtuoso Analog Design Environment GXL User Guide

### High Yield Estimation

**Note:** If you opened the High Yield Estimation form by clicking the *Simulation Options* button on the Run toolbar, click the *Run Simulation* button on the Run toolbar to start the high yield estimation run.

The Run Log | Log File Viewer form appears displaying information regarding the progress of the initial Monte Carlo Sampling run, the yield estimate at each iteration, and the summary of the High Yield Estimation run.

The screenshot shows the 'Run Log | Log file viewer <2>' window from Cadence. The window title is 'Run Log | Log file viewer <2>'. The menu bar includes 'File', 'View', and 'Help'. The Cadence logo is in the top right corner. The main content area displays the following text:

```
Starting high yield estimation for following spec corners:  
SlewRate.Slew_Rate  
  
Estimation method: Worst Case Distance  
  
Spec corner SlewRate.Slew_Rate  
Yield result of Monte Carlo Sampling: 100%  
Statistical variable reduction -  
Total number of statistical variables: 178  
Number of statistical variables after reduction: 11  
Statistical variables selected:  
Mean Stddev  
statistical:global:random13 0 1  
statistical:mismatch:amp.M10:rn1_18 0 1  
statistical:mismatch:amp.M10:rn2_18 0 1  
statistical:mismatch:amp.M11:rn1_18 0 1  
statistical:mismatch:amp.M12:rp1_18 0 1  
statistical:mismatch:amp.M1:rn1_18 0 1  
statistical:mismatch:amp.M1:rn2_18 0 1  
statistical:mismatch:amp.M4:rp1_18 0 1  
statistical:mismatch:amp.M4:rp2_18 0 1  
statistical:mismatch:amp.M5:rp1_18 0 1  
statistical:mismatch:amp.M5:rp2_18 0 1  
  
Initial estimate based on Monte Carlo Sampling -  
Predicted WCD point:  
statistical:global:random13 -0.601658  
statistical:mismatch:amp.M10:rn1_18 0.801355  
statistical:mismatch:amp.M10:rn2_18 0.751594  
statistical:mismatch:amp.M11:rn1_18 0.702289  
statistical:mismatch:amp.M12:rp1_18 1.4343  
statistical:mismatch:amp.M1:rn1_18 -1.35984  
statistical:mismatch:amp.M1:rn2_18 -1.52196  
statistical:mismatch:amp.M4:rp1_18 0.957519  
statistical:mismatch:amp.M4:rp2_18 0.794871  
statistical:mismatch:amp.M5:rp1_18 -2.65109  
statistical:mismatch:amp.M5:rp2_18 -1.13385  
  
Predicted WCD: 4.26  
Iteration 1 -
```

The results for the run are displayed in the Results tab, as shown below.



Name	Yield in Sigma	Yield in Percentage	MC Yield	Target	Sigma to Target	Status
<b>s: Yield Estimation by Worst Case Distance Method</b>						
<u>_Stage_Opamp:OpAmp_AC_top:1</u>						
Current(summary)		100	5.65206			
urrent	5.95154	99.99999973	100	< 1.5m	5.65206	converged after 2 iterations
Gain(summary)		98.5	2.4062			
ain	2.49472	98.73942853	98.5	> 56.8	2.4062	converged after 3 iterations
UGF(summary)		97	2.00863			
GF	2.57539	98.99872712	97	> 170M	2.00863	converged after 2 iterations
<u>_Stage_Opamp:OpAmp_TRAN_top:1</u>						
SettlingTime(summary)		100	11.8757			
ettlingTime	9.53399	100.000000000	100	< 7n	11.8757	converged after 5 iterations
Swing(summary)		100	3.61876			
wing	4.09053	99.99569621	100	> 1.38	3.61876	converged after 2 iterations

The Results tab shows the following columns:

- **Name:** Displays the name of specification.
- **WCD:** Displays the WCD value for the given specification.
- **Yield in Sigma:** Displays the yield value in sigma. This value is calculated as shown below:

$\text{Yield in Sigma} = \sqrt{2} * \text{erfinv}(\text{Yield in Percentage}/100)$

where, erfinv is the inverse error function.

If the yield in sigma is greater than 8.2, the yield in percentage is displayed as 100%.

- **Yield in Percentage:** Displays the yield value in percentage. This value is calculated as shown below:

$\text{Yield in Percentage} = 0.5 + 0.5 * \text{erf}(\text{WCD}/\sqrt{2})$

where, erf is the error function.

**Note:** The yield in percentage value is by default displayed with 10 digits. To change the number of digits to be displayed for this value, set the value of variable `digitsToShowForYieldInPercentage` environment variable. You can display a maximum of 53 digits for these values.

- **MC Yield:** Displays the yield value from the Monte Carlo run.
- **Target:** Displays the target to be achieved for the given specification.
- **Sigma to Target:** Displays the sigma to target value.

- *Status*: Displays the convergence status for each specification.

The tool uses the following two convergence criteria:

- The predicted WCD point is on the specification boundary (within tolerance < 0.02).  
The log file reports the 'Spec value error' at each iteration.

```
spec_value_error = abs(spec_value - spec_target) / abs(nominal spec_value - spec_target)
```

- The angle between the gradient vector and the statistical variable vector is < 8 deg.  
The log file reports the 'Gradient direction error' at each iteration.

The convergence status can be any of the following:

---

Status	Description
<i>converged after x iterations</i>	<p>The yield estimate converged after <i>x</i> iterations.</p> <p>For example, the yield estimate for the <code>Slew_rate</code> specification converged after 2 iterations.</p>
<i>lower boundary / least error WCD, did not converge after x iterations</i>	<p>The yield estimate did not converge after <i>x</i> iterations, where <i>x</i> is the maximum number of iterations specified in the <a href="#">High Yield Estimation options form</a>. This is either because the specification is non-linear or the specification boundary is far away from the nominal point.</p> <p>When the specification is non-linear and the spec boundary is far away from the nominal point, the lower boundary WCD error is reported, where WCD is the value from the final iteration.</p> <p>When the specification is linear and the spec boundary or the WCD boundary is far away from the nominal point, the least error WCD among a specific number of iterations is reported, where WCD is the value from any of the iterations.</p> <p>Note that for every iteration, WCD can increase by a maximum of six sigma.</p>

---

Status	Description
<i>estimate based on MC data / lower boundary / least error WCD, stopped because of error evaluating WCD point sensitivity on iteration x</i>	<p>The application of the WCD method on the specification is stopped before reaching the maximum number of iterations because of a simulation or measurement evaluation error in evaluating the WCD point sensitivity.</p> <p>The lower boundary is reported if it is identified. Else, the WCD value with least error among iterations is reported.</p>
<i>estimate based on MC data / lower boundary / least error WCD, stopped because of error evaluating WCD point on iteration x</i>	<p>The application of the WCD method on the specification is stopped before reaching the maximum number of iterations because of a simulation or measurement evaluation error in evaluating the WCD candidate point.</p> <p>If the process was stopped on the first iteration, the estimate based on Monte Carlo results is reported. Else, a low boundary, if identified, or the WCD value with least error among iterations is reported.</p>
<i>skipped because of low Monte Carlo yield</i>	No iteration was run because the initial Monte Carlo run reported low yield for the specification.

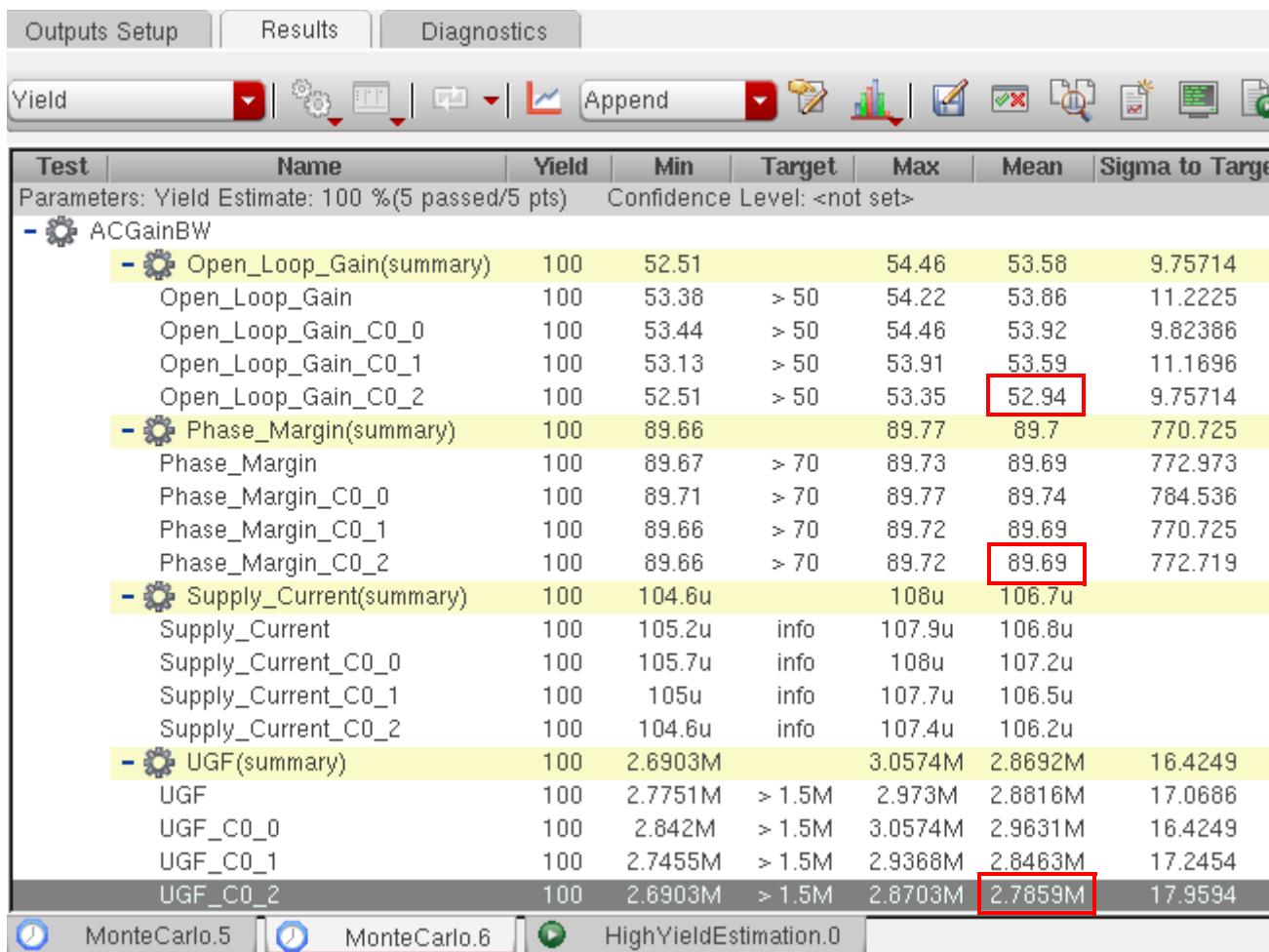
## High Yield Estimation with Multiple Corners

If you enable multiple corners for simulation, Monte Carlo simulation is performed for all of the enabled corners. However, for high yield estimation, only a single corner is selected for each specification. The selected corner is the one that has the worst mean value out of all the corners in the Monte Carlo results for a given specification.

Consider an example of the Monte Carlo results for a test with multiple corners, as shown below.

# Virtuoso Analog Design Environment GXL User Guide

## High Yield Estimation



In this case, the worst mean value for each specification is for corner C0\_2. As a result, for each specification, the tool performs high yield estimation by using corner C0\_2.

After the worst case distance analysis is complete, the yield in sigma, yield in percentage, and convergence status are shown only for those corners that were used to perform the high yield estimation, as shown below.

# Virtuoso Analog Design Environment GXL User Guide

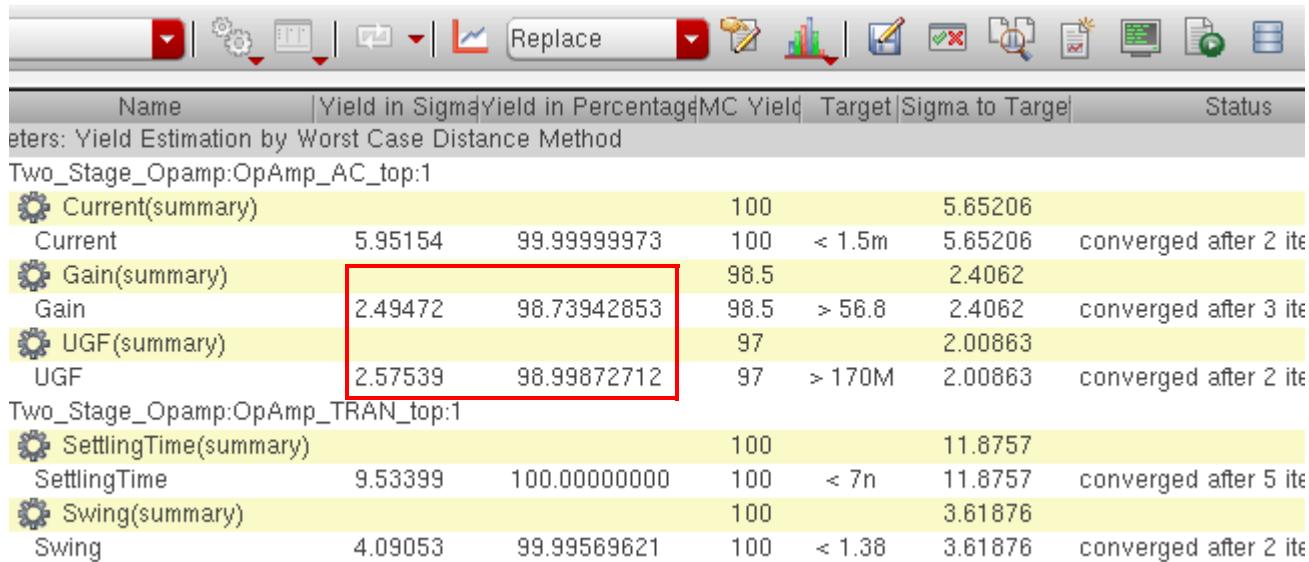
## High Yield Estimation

Name	Yield in Sigma	Yield in Percentage	MC Yield	Target	Sigma to Target	Status
Process: Yield Estimation by Worst Case Distance Method						
CGainBW						
Open_Loop_Gain(su...)	100		9.75714			
Open_Loop_Gain	100	> 50	11.2225			
Open_Loop_Gain_C0_0	100	> 50	9.82386			
Open_Loop_Gain_C0_1	100	> 50	11.1696			
Open_Loop_Gain_C0_2	6.08148	99.99999988	100	> 50	9.75714	converged after 9 iterations
Phase_Margin(summ...)	100		770.725			
Phase_Margin	100	> 70	772.973			
Phase_Margin_C0_0	100	> 70	784.536			
Phase_Margin_C0_1	100	> 70	770.725			
Phase_Margin_C0_2	23.8259	100.000000000	100	> 70	772.719	least error WCD, did not converge

## Creating Statistical Corners From a Worst Case Distance Analysis

After running the high yield estimation run, if you find that the required yield value is not achieved for any specification, you can create statistical corner based on the worst case distance point for that specification. You can use these new corners to run optimization or manually tune the design accordingly.

For example, if you run high yield estimation for a design without skipping the specs that have monte carlo yield of less than 3 sigma, the results might show specifications that have yield value less than 3 sigma, as shown below:

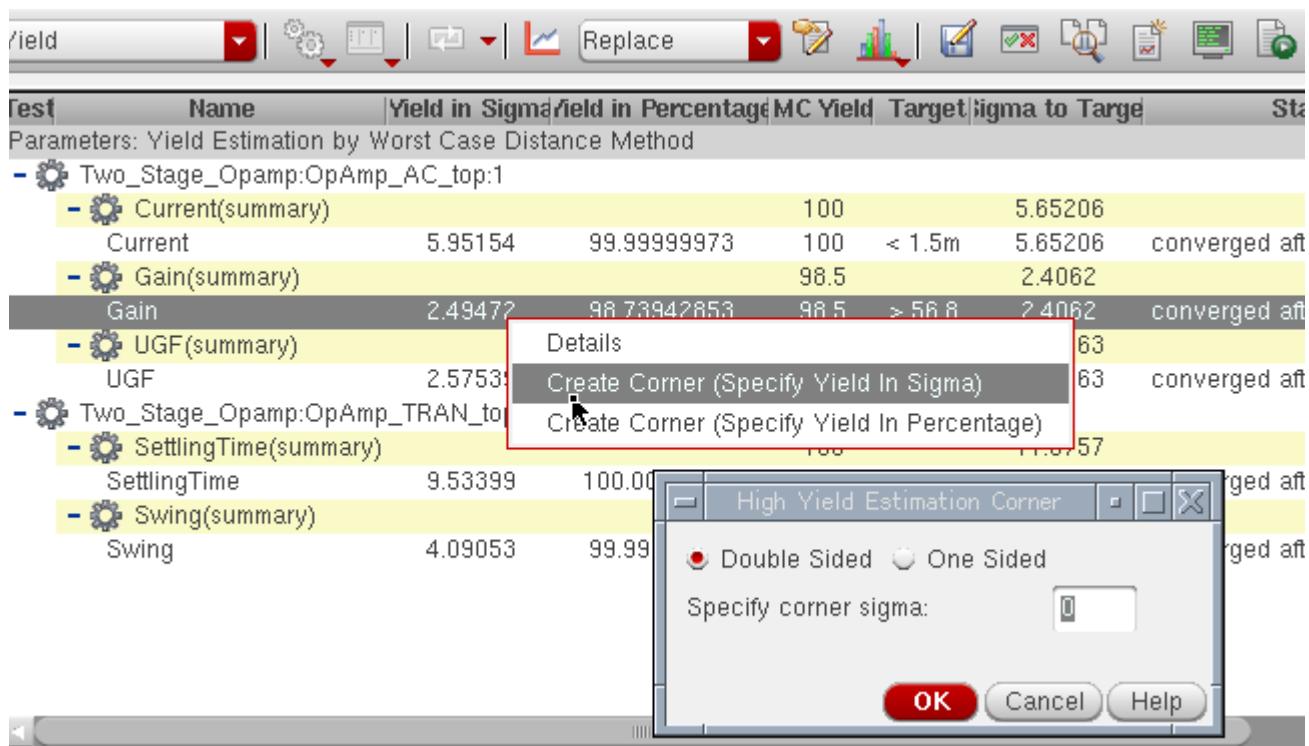


Name	Yield in Sigma	Yield in Percentage	MC Yield	Target	Sigma to Target	Status
Parameters: Yield Estimation by Worst Case Distance Method						
Two_Stage_Opamp:OpAmp_AC_top:1						
Current(summary)			100		5.65206	
Current	5.95154	99.99999973	100	< 1.5m	5.65206	converged after 2 ite
Gain(summary)			98.5		2.4062	
Gain	2.49472	98.73942853	98.5	> 56.8	2.4062	converged after 3 ite
UGF(summary)			97		2.00863	
UGF	2.57539	98.99872712	97	> 170M	2.00863	converged after 2 ite
Two_Stage_Opamp:OpAmp_TRAN_top:1						
SettlingTime(summary)			100		11.8757	
SettlingTime	9.53399	100.000000000	100	< 7n	11.8757	converged after 5 ite
Swing(summary)			100		3.61876	
Swing	4.09053	99.99569621	100	< 1.38	3.61876	converged after 2 ite

Note that in the above results, the monte carlo yield for Gain and UGF is less than 3 sigma. To improve the performance for these specifications, you can create a statistical corner based on the Worst Case Distance (WCD) point of these specifications. For this, you can specify a target yield value greater than three sigma, either in terms of sigma or in terms of percentage value.

To create a WCD corner based on yield in sigma, right-click on a specification and choose *Create Corner (Specify Yield In Sigma)*.

The High Yield Estimation Corner form appears as shown below:



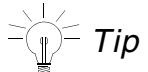
In the High Yield Estimation Corner form, do the following:

1. Select one of the following two values to specify the yield representation:
  - Double sided*: Specifies that yield in sigma is represented as probability integration from  $-x$  sigma to  $+x$  sigma in the Gaussian distribution, where  $x$  is the target sigma value given in the *Specify corner sigma* field. This is the default value.
  - One sided*: Specifies that yield in sigma is represented as probability integration from negative infinite to  $+x$  sigma in the Gaussian distribution, where  $x$  is the target sigma value given in the *Specify corner sigma* field.
2. In the *Specify corner sigma* field, specify a target sigma value to which you want to extend the WCD point.
3. Click *OK*.

A statistical corner is created by using the specification values that can improve the circuit yield to the target sigma value. The name of the statistical corner is prefixed with `WCD_`.

The new corner is added to the Data View and the **Corners Setup** form.

Alternatively, you can choose *Create Corner (Specify Yield In Percentage)* and specify the target yield value to be achieved in terms of percentage.

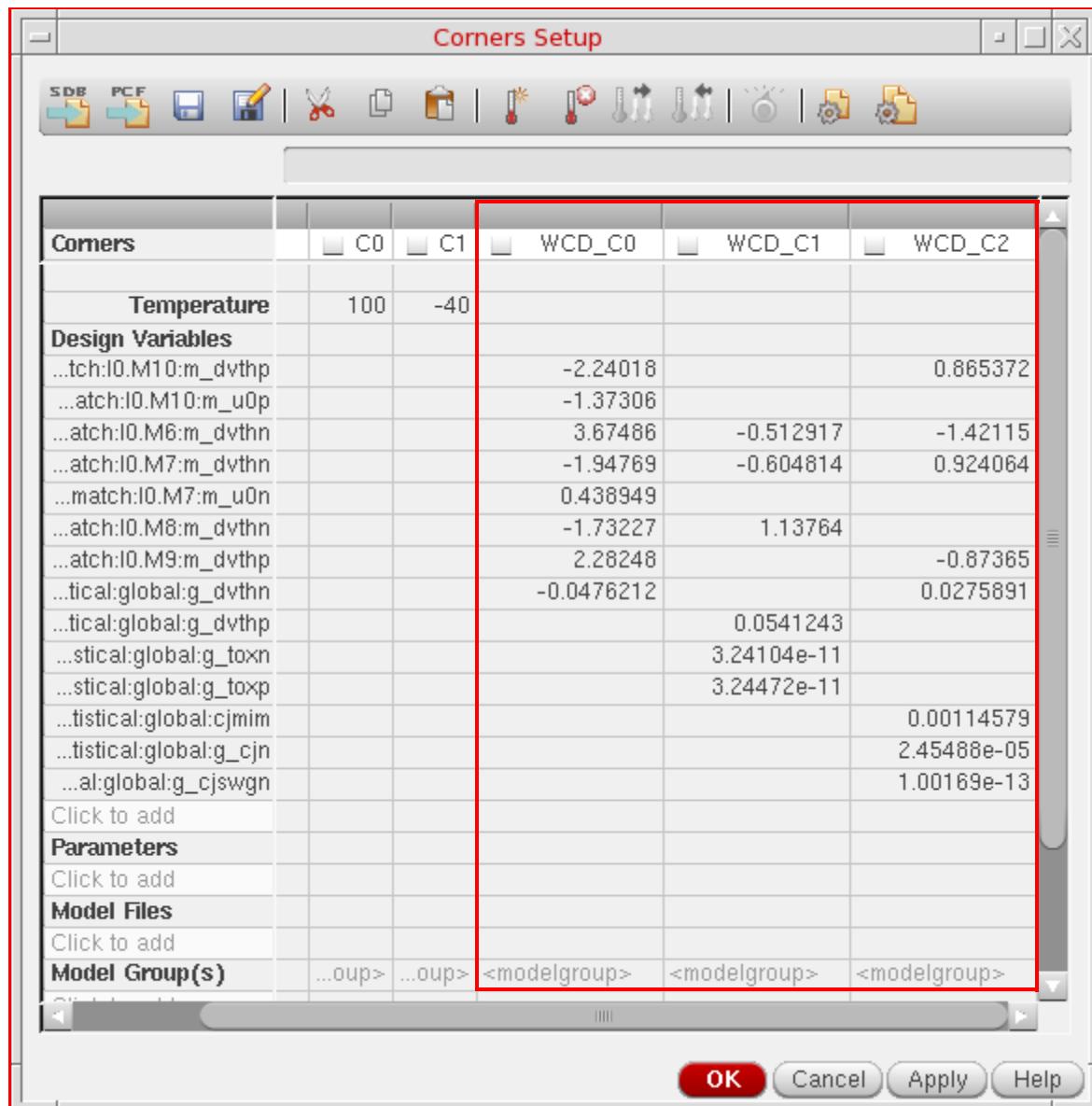


**Tip**  
For a normal design, you can set for a corner sigma value equal to 3.0. However, for a high yield design, you can set a corner sigma value equal to 6.0. A good practice is to increase the sigma value by 0.5 to 1.0 in each iteration instead of increasing it by large amount.

## Virtuoso Analog Design Environment GXL User Guide

### High Yield Estimation

In the example shown above, let us create three statistical corners by setting the corner sigma for Current, Gain, and UGF equal to 6, 3, and 3, respectively. In this case, the corners are created as shown below:

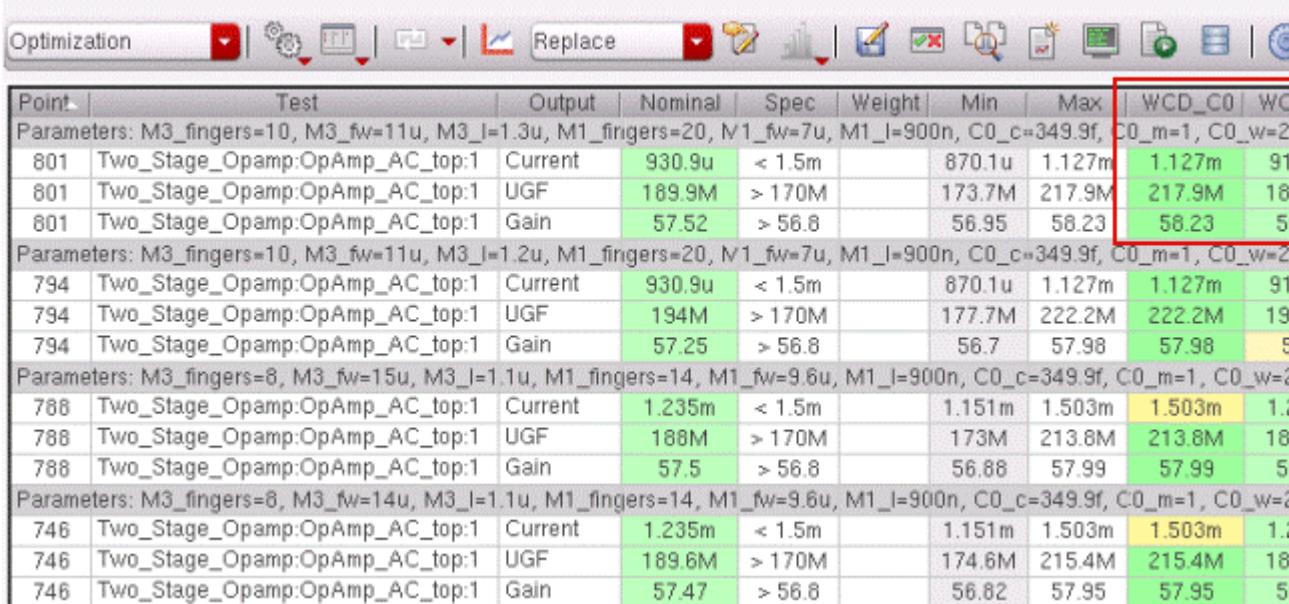


## Virtuoso Analog Design Environment GXL User Guide

### High Yield Estimation

---

Now, if you enable corners and run optimization on the design, the results for the new statistical corners appear as shown below:



Point	Test	Output	Nominal	Spec	Weight	Min	Max	WCD_C0	WCI
Parameters: M3_fingers=10, M3_fw=11u, M3_l=1.3u, M1_fingers=20, M1_fw=7u, M1_l=900n, C0_c=349.9f, C0_m=1, C0_w=20									
801	Two_Stage_Opamp:OpAmp_AC_top:1	Current	930.9u	< 1.5m		870.1u	1.127m	1.127m	91
801	Two_Stage_Opamp:OpAmp_AC_top:1	UGF	189.9M	> 170M		173.7M	217.9M	217.9M	185
801	Two_Stage_Opamp:OpAmp_AC_top:1	Gain	57.52	> 56.8		56.95	58.23	58.23	56
Parameters: M3_fingers=10, M3_fw=11u, M3_l=1.2u, M1_fingers=20, M1_fw=7u, M1_l=900n, C0_c=349.9f, C0_m=1, C0_w=20									
794	Two_Stage_Opamp:OpAmp_AC_top:1	Current	930.9u	< 1.5m		870.1u	1.127m	1.127m	91
794	Two_Stage_Opamp:OpAmp_AC_top:1	UGF	194M	> 170M		177.7M	222.2M	222.2M	193
794	Two_Stage_Opamp:OpAmp_AC_top:1	Gain	57.25	> 56.8		56.7	57.98	57.98	51
Parameters: M3_fingers=8, M3_fw=15u, M3_l=1.1u, M1_fingers=14, M1_fw=9.6u, M1_l=900n, C0_c=349.9f, C0_m=1, C0_w=20									
788	Two_Stage_Opamp:OpAmp_AC_top:1	Current	1.235m	< 1.5m		1.151m	1.503m	1.503m	1.2
788	Two_Stage_Opamp:OpAmp_AC_top:1	UGF	188M	> 170M		173M	213.8M	213.8M	187
788	Two_Stage_Opamp:OpAmp_AC_top:1	Gain	57.5	> 56.8		56.88	57.99	57.99	56
Parameters: M3_fingers=8, M3_fw=14u, M3_l=1.1u, M1_fingers=14, M1_fw=9.6u, M1_l=900n, C0_c=349.9f, C0_m=1, C0_w=20									
746	Two_Stage_Opamp:OpAmp_AC_top:1	Current	1.235m	< 1.5m		1.151m	1.503m	1.503m	1.2
746	Two_Stage_Opamp:OpAmp_AC_top:1	UGF	189.6M	> 170M		174.6M	215.4M	215.4M	189
746	Two_Stage_Opamp:OpAmp_AC_top:1	Gain	57.47	> 56.8		56.82	57.95	57.95	56

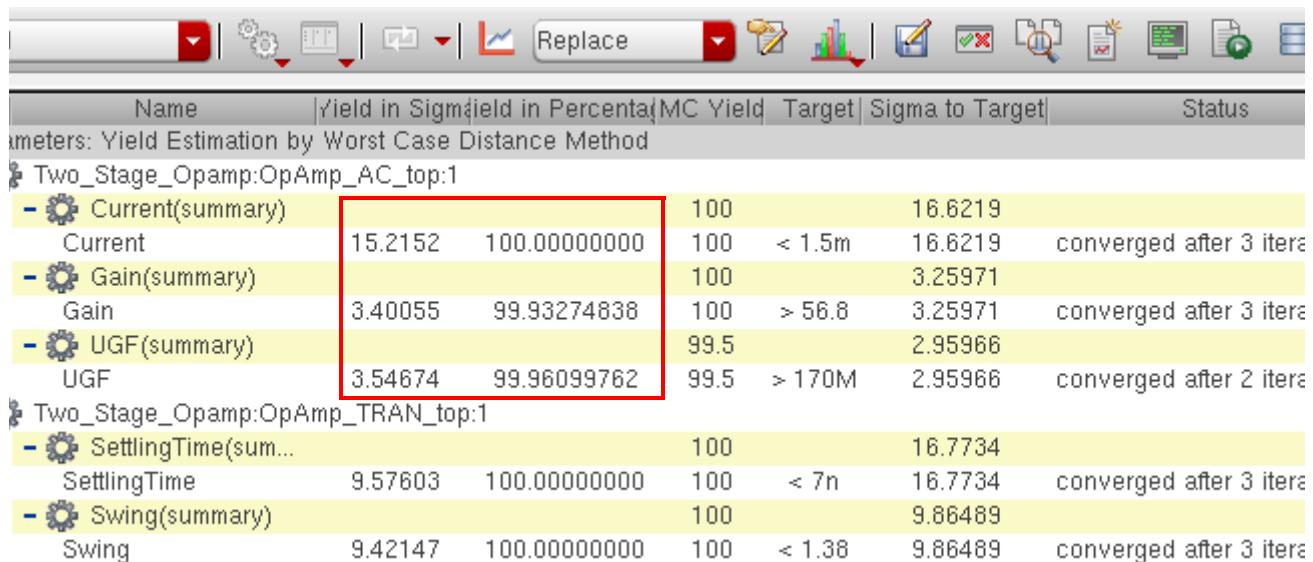
From the results of an optimization run, you can identify a best design point and create a reference point. For this, right-click on the design point and choose *Create Reference Point*.

Next, you can use this reference point and run high yield estimation to verify if there is any improvement in the yield for the selected specifications. To use the reference point for high yield estimation run, ensure that the *Use Reference Point* check box on the High Yield Estimation form is selected.

## Virtuoso Analog Design Environment GXL User Guide

### High Yield Estimation

For the example given above, the results show significant improvement in the yield and the worst case distance values, as shown below:



Name	/yield in Sigma	/yield in Percent	MC Yield	Target	Sigma to Target	Status
Parameters: Yield Estimation by Worst Case Distance Method						
Two_Stage_Opamp:OpAmp_AC_top:1						
- Current(summary)						
Current	15.2152	100.000000000	100	< 1.5m	16.6219	converged after 3 itera
- Gain(summary)						
Gain	3.40055	99.93274838	100	> 56.8	3.25971	converged after 3 itera
- UGF(summary)						
UGF	3.54674	99.96099762	99.5	> 170M	2.95966	converged after 2 itera
Two_Stage_Opamp:OpAmp_TRAN_top:1						
- SettlingTime(summary)						
SettlingTime	9.57603	100.000000000	100	< 7n	16.7734	converged after 3 itera
- Swing(summary)						
Swing	9.42147	100.000000000	100	< 1.38	9.86489	converged after 3 itera

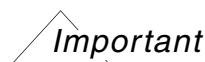
You can see that the yield in sigma value has increased for all Current, Gain, and UGF as compared to the results of the [previous](#) high yield estimation run.

---

## Multi-Technology Simulation

---

You can use multi-technology simulation (MTS) when designing high-speed interfaces between integrated circuits (ICs) to see the effects of signals traveling between two or more chips. You can also use MTS for custom IC system-in-package (SiP) designs. SiP designs are fully functional systems or subsystems in an IC package. MTS allows you to combine passive components with ICs fabricated using different process technologies while minimizing parasitic loading.



In order to run a simulation in multi-technology mode, your design must have a [design configuration view](#). This requirement makes it possible for you to configure MTS blocks from the [hierarchy editor](#) (HED).

Your MTS design should consist of a top-level schematic that contains two or more blocks, each with its own associated set of process definitions, model libraries, simulation states, and other setup files. You cannot use MTS with a non-hierarchical schematic because each different technology must be self-contained. You must instantiate device instances from different technologies in different cell views so that the program can map model information appropriately.

To specify MTS, you need to do the following:

- [Enable multi-technology simulation](#) on the Choosing Simulator form.
- [Specify MTS options](#).

See also

- [Specifying Test and Block Information when Adding Model Files to Corners](#) on page 192
- [Disabling MTS for Third-Party Simulators](#) on page 192

## Enabling Multi-Technology Simulation



In order to run a simulation in multi-technology mode, your design must have a design configuration view.

To enable multi-technology simulation, do the following:

1. On the Data View pane, right-click the test or analysis name and select *Simulator*.  
The Choosing Simulator form appears.



2. Mark the *Multi-Technology Mode* check box.  
3. Click *OK*.

You can now specify MTS options.

## Specifying MTS Options



Before specifying MTS options, you must [enable MTS](#).

To specify MTS options, do the following:

- On the Tests and Analyses pane, [right-click the test or analysis name](#) and select *MTS Options*.

**Note:** The test or analysis you right-click must be one for which you have enabled MTS.

The MTS Options form appears.

Library	Cell	MTS_BLOCK	modelFiles	scale	scalem	temp	tnom
analogLib	cap	<input type="checkbox"/>	...				
analogLib	idc	<input type="checkbox"/>	...				
analogLib	res	<input type="checkbox"/>	...				
analogLib	vdc	<input type="checkbox"/>	...				
analogLib	vpwl	<input type="checkbox"/>	...				
demoLib	adc_cascode_opamp_sim	<input type="checkbox"/>	...				
ether_adcflash_RAD90	adc_cascode_opamp	<input type="checkbox"/>	...				
gpdk090	nmos2v	<input type="checkbox"/>	...				
gpdk090	pmos2v	<input type="checkbox"/>	...				

MTS blocks are indicated by a check mark in the *MTS\_BLOCK* column.  
You can specify settings that apply to particular cells or [instances](#) on this form.

See the following topics for more information:

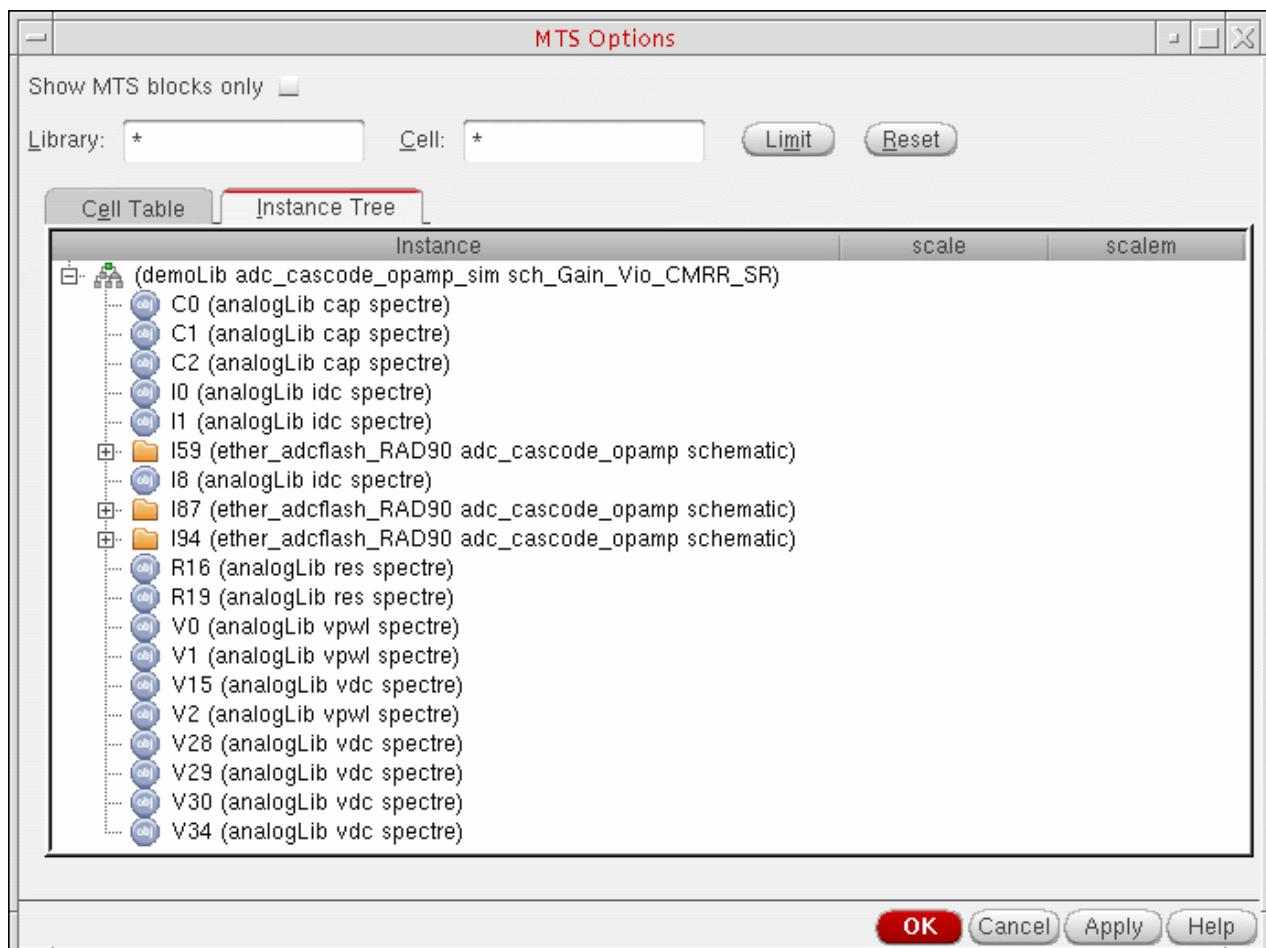
- [Viewing Cells by Instance Name](#) on page 188
- [Limiting the View by Library or Instance](#) on page 190
- [Specifying a Cell as an MTS Block](#) on page 190
- [Specifying Simulator Options for an MTS Block](#) on page 191
- [Specifying a Model Library for an MTS Block](#) on page 191

## Viewing Cells by Instance Name

To view cells by instance name, do the following:

- On the MTS Options form, select the *Instance Tree* tab.

Instance names appear alphabetically in a tree view on the form.



To restore the cell table, do the following:

- Select the *Cell Table* tab.

The cell table view appears on the form.

See also [“Limiting the View by Library or Instance”](#) on page 190.

## **Limiting the View by Library or Instance**

You can limit what appears on the MTS Options form by library name, cell name, or both.

**Note:** You can reset the filter to view all cells in all libraries by clicking *Reset*.

To limit the view of design components on the MTS Options form to only those in a particular library or set of libraries, do the following:

1. In the *Library* field, type a filtering string.
2. Click *Limit*.

Only those design libraries that match the filtering string appear on the MTS Options form.

To limit the view of design components on the MTS Options form to a particular cell or set of cells, do the following:

1. In the *Cell* field, type a filtering string.
2. Click *Limit*.

Only those cells that match the filtering string appear on the MTS Options form.

To limit the view by library name and by cell name, do the following:

1. In the *Library* field, type a filtering string.
2. In the *Cell* field, type a filtering string.
3. Click *Limit*.

Only those cells that match the filtering string in the *Cell* field contained in those libraries that match the filtering string in the *Library* field appear on the MTS Options form.

## **Specifying a Cell as an MTS Block**

To specify a cell as an MTS block, do the following:

- In the row for that cell, mark the *MTS\_BLOCK* check box.

The library cell you selected is enabled for multi-technology simulation.

If you open the Model Library Setup form, you will see a new model library tree for that library and cell so that you can specify model libraries specific to that MTS block.

## Specifying Simulator Options for an MTS Block

To specify simulator options (such as *t<sub>nom</sub>*, *scale*, *temp*, or *scalem*) for a particular cell or instance that you have specified as an MTS block, do the following:

- Click in the column for the simulator option you want to specify and type a value.  
The value you specify applies to the simulation of that MTS block.

## Specifying a Model Library for an MTS Block

To specify a model library for an MTS block, do the following:

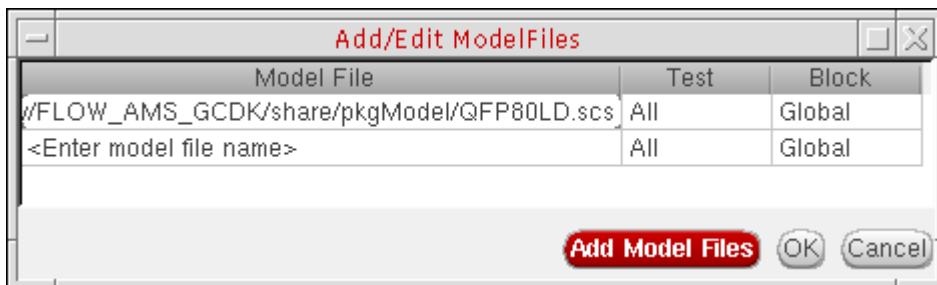
- In the row for the cell that you have specified as an MTS block, click the button in the *modelFiles* column.  
The Model Library Setup form appears. See “[Specifying Model Libraries](#)” in the *Virtuoso Analog Design Environment XL User Guide* for more information about using this form.

## Specifying Test and Block Information when Adding Model Files to Corners

To specify a particular test to which you want to apply a model file or a particular section of a model file when adding model files for corners analysis, do the following:

1. Follow the instructions for adding model files to corners but do not click *OK*.

The model file you added appears on the Add/Edit Model Files form.



2. In the *Test* column, double-click and, from the drop-down menu, select a particular test for which you want to use the model file.

By default, the program uses the model file for all tests.

3. In the *Block* column, double-click and, from the drop-down menu, select a particular MTS block to which you want to apply the model file.

By default, the program uses the model library for the entire design: *Global*.

4. Click *OK*.

## Disabling MTS for Third-Party Simulators

To disable the MTS feature for a third-party simulator that does not support it, do the following to overload the method:

- Change

```
defmethod( asiIsMTSSupported ( ( session simulation_session ) )
          t
        )
to
defmethod( asiIsMTSSupported ( ( session className ) )
          nil
        )
```

## **Virtuoso Analog Design Environment GXL User Guide**

### Multi-Technology Simulation

---

where, *className* is the name of the third-party simulator class derived from simulation\_session.

## Detecting Context Symbol/Name Collisions

As a context developer, you need to check for conflicts that arise when you define a function or variable in more than one context (name collisions).

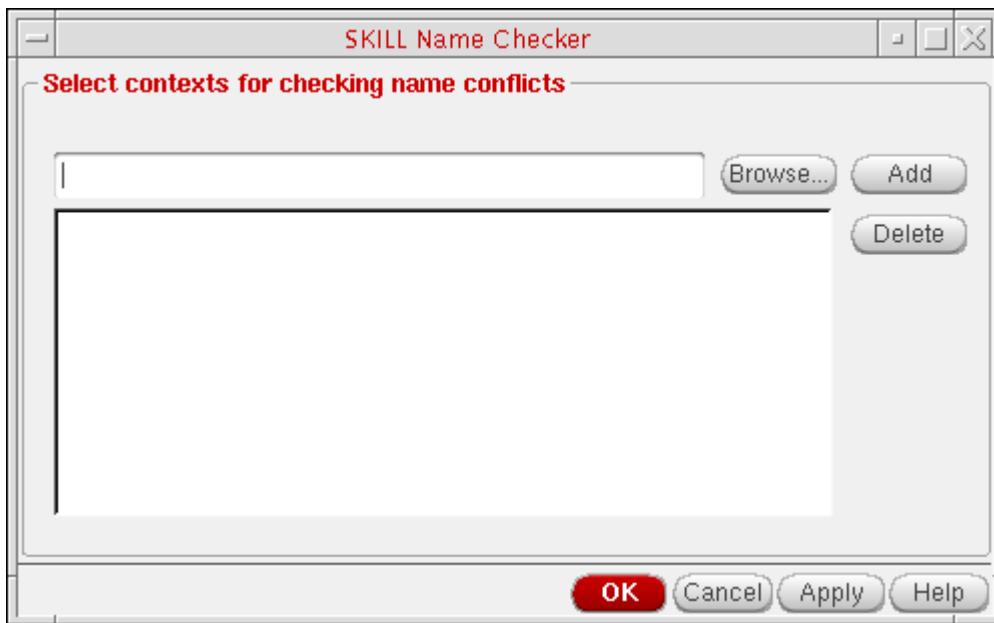
To check for name collisions in loaded contexts, do the following:

1. In the CIW, choose *Tools – Conversion Tool Box*.

The Conversion Tool Box appears.

2. Click *Check SKILL Name Conflicts*.

The SKILL Name Checker form appears.



3. Click *Browse*.

A browser window appears.

4. For each context file you want to add to the check, do the following:

- a. Navigate to and select a valid context file and click *Apply*.

The file name appears in the field on the SKILL Name Checker form.

- b. On the SKILL Name Checker form, click *Add*.

The file name moves from the field to the list area.

5. On the SKILL Name Checker form, click *OK*.

## Virtuoso Analog Design Environment GXL User Guide

### Multi-Technology Simulation

---

Duplicate definitions appear in the SKILL Name Checker Results window as follows:

symbol '*functionName* already defined by context "*contextName*"

where *functionName* is the name of the function and *contextName* is the first context in which the program found *functionName*.

**Virtuoso Analog Design Environment GXL User Guide**  
Multi-Technology Simulation

---

---

## Design Characterization and Modeling

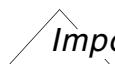
---

You can use the Virtuoso® Design Characterization and Modeling (DCM) feature of the Analog Design Environment GXL (ADE GXL) to create the following:

- Top-down Verilog-A[MS] and VHDL-AMS behavioral models
- Bottom-up Verilog-A[MS], VHDL-AMS, Verilog-D, and Liberty behavioral models

Using DCM, you can verify a design by simulating a design over a number of conditions and corners and comparing measured behavior to minimum and maximum limits. You can also just view measured results (without specifying a minimum or maximum). DCM creates tests, parameters, sweeps, corners, and SKILL scripts that you can use in ADE XL. The program writes test information to a state directory containing all the required state files.

Verilog-A[MS] enables high-performance simulation with SPICE accuracy for greater than an order of magnitude improvement in performance. Behavior of Verilog-A[MS] digital models can change with environment (temperature and supply voltage), input stimulus (slope), and connection change or parasitics (load). Other analog parameters can be swept for different functional cell types.



DCM also includes the DCM Developer tool for creating cell types. For more information, see [Cell Type Development](#).

See the following topics for more information:

- [DCM Process Flows](#)
- [Setting Up VHDL-AMS and Other Model Formats](#)
- [Launching DCM](#)
- [Accessing the Start Form](#)
- [Saving Changes](#)
- [Using the Tools Menu](#)

## **Virtuoso Analog Design Environment GXL User Guide**

### Design Characterization and Modeling

---

- [Viewing the Last Error](#)
- [Viewing Generated Files](#)
- [Exiting Design Characterization and Modeling](#)

**Note:** DCM does not support the `-nograph` command-line option for Virtuoso workbenches.

## DCM Process Flows

The process flows for using DCM are as follows:



***Some steps may be optional, depending on what you want to accomplish using DCM.***

### Bottom-Up Modeling

1. [Launching Bottom-Up Modeling](#)
2. [Specifying a Design](#)
3. [Specifying Modeling Requirements](#)
  - a. [Specifying Bottom-Up Modeling for AMS Models](#)
  - b. [Specifying Bottom-Up Modeling for Verilog-D](#)
  - c. [Viewing Bottom-Up Liberty Generated Files](#)
4. [Setting Up Model Calibration Requirements](#)
  - a. [Calibrating Models](#)
  - b. [Modifying Calibration Sweep Parameters](#)
  - c. [Specifying Parameters](#)
  - d. [Specifying Options](#)
5. [Generating Models and Running Tests](#)
  - a. [Generating Tests and Models](#)
  - b. [Running Tests](#)

### Top-Down Modeling

1. [Launching Top-Down Modeling](#)
2. [Specifying a Function](#)
3. [Specifying Top-Down Modeling for AMS Models](#)

**4. Setting Up Model Calibration Requirements**

- a. Calibrating Models**
- b. Modifying Calibration Sweep Parameters**
- c. Specifying Parameters**
- d. Specifying Options**

**5. Generating Tests and Models**

## **Liberty MS Model Generation**

- 1. Launching Black Box Liberty .lib MS Modeling**
- 2. Specifying a Design**
- 3. Specifying a Function**
- 4. Adding Model Components**
- 5. Generating the Model**

## **Verilog-A[MS] Model Calibration**

- 1. Specifying the Source Model**
- 2. Setting Up the Model**
- 3. Specifying the Destination Model**
- 4. Saving the Model Calibration Setup**
- 5. Calibrating the Model**

## **DCM Developer**

- 1. Launching dcmDeveloper to Create New Cell Types**
- 2. Adding and Configuring Test Benches**
- 3. Loading State Files**
- 4. Adding Models**
- 5. Specifying Cell Type Generation Options**

- 6. Specifying the Output Destination**
- 7. Saving the dcmDeveloper Data**

## **Design Verification**

- 1. Launching Complete Modeling**
- 2. Specifying a Design**
- 3. Enabling Design Verification**
- 4. Defining Specifications for Design Verification**
- 5. Specifying Conditions for Design Verification**

## Setting Up VHDL-AMS and Other Model Formats

DCM supports VHDL-AMS and other externally-defined model formats. Support for these additional model formats is controlled by:

- Entries within the create block of the cell type .gui file
- A configuration file (.dmi) which is referenced in the .gui create block



***VHDL-AMS in AMS Designer does not yet natively support table models (\$table\_model). As a result, Verilog-A[MS] modules and instances are created with the table call for Bottom-Up calibrated VHD-AMS models. For a description of the imposed limitations, see [Analog Design Environment GXL Known Problems and Solutions](#).***

### Setting Up the Create Block in the .gui File

To support VHDL-AMS and other external model formats, specify the following lines in the create block in the .gui file:

#### Syntax

```
Top-Down <formatDesc> (<templateFileName>) <configFileName>
<formatDesc> (<templateFileName>) <configFileName>
```

Where the first line defines the information for the top-down flow, and the line not annotated with “Top-Down” controls the bottom-up flow.

#### Parameters

**Table 6-1 Parameters for External Model Format in Create Block**

Parameter	Description
formatDesc	The description of the model format to be added as an option in the Options pane of the DCM GUI. This option controls the enabling of this model.
templateFileName	The template file within the simulator subdirectory.

**Table 6-1 Parameters for External Model Format in Create Block**

Parameter	Description
configFileName	The name of the .dmi configuration file. DCM searches for this file based on the setting of the <u>AXL_DCM_GUI_EXTRAS_RULES</u> environment variable.

### Example

```
VHDL-AMS (buf.vhms) vhms  
Top-down VHDL-AMS (top_down.vhms) vhms
```

### Specifying Environment Variables

In order to utilize external model formats, you must specify the following environment variables:

**Table 6-2 Environment Variables for External Model Formats**

Environment Variable	Description
AXL_DCM_GUI	Set this environment variable to the directory name where the cell types are stored.  Set either this variable or AXL_DCM_GUI_DEF. If this variable is set, the default installed cell types and the ones in the defined directories are made available.
AXL_DCM_GUI_DEF	Set this environment variable to the directory name where the cell types are stored.  Set either this variable or AXL_DCM_GUI. If this variable is set, only cell types from the defined directories are made available.
AXL_DCM_GUI_EXTRAS	Set this environment variable to true to enable the GUI flow for supporting external model formats.
AXL_DCM_GUI_EXTRAS_RULES	The location of the .dmi files.

## Launching DCM

To launch Virtuoso® Design Characterization and Modeling (DCM) from the Command Interpreter Window (CIW), do the following:

- Choose one of the following:
  - Tools – Characterization and Modeling – Complete*
  - Tools – Characterization and Modeling – Top Down Modeling*
  - Tools – Characterization and Modeling – Bottom Up Modeling*
  - Tools – Characterization and Modeling – Design Verification*
  - Tools – Characterization and Modeling – Black Box Liberty .lib MS*

Your selection determines which component tabs appear in the Virtuoso Design Characterization and Modeling window as follows:

---

**Tabs that appear in the Virtuoso Design Characterization and Modeling window**

Submenu Item	<u>Design</u>	<u>Function</u>	<u>Top-Down</u>	<u>Bottom-Up</u>	<u>Design Verification</u>	<u>Parameters</u>	<u>Options</u>	<u>Black Box</u>
<i>Complete</i>	X	X	X	X	X	X	X	
<i>Top Down Modeling</i>		X	X					
<i>Bottom Up Modeling</i>	X	X		X		X	X	
<i>Design Verification</i>	X	X			X	X	X	
<i>Black Box Liberty .lib MS</i>	X	X						X

The Virtuoso Design Characterization and Modeling Start form appears.

Alternatively, you can launch DCM from the ADE XL Intro Wizard:

1. In the CIW, choose *Tools – ADE XL*.
- The ADE XL Intro Wizard form appears.
2. On the ADE XL Intro Wizard form, select *Launch DCM*.
3. Click *OK*.

The Virtuoso Design Characterization and Modeling Start form appears.

To begin generating behavioral models, see Chapter 7, “Model Generation.”

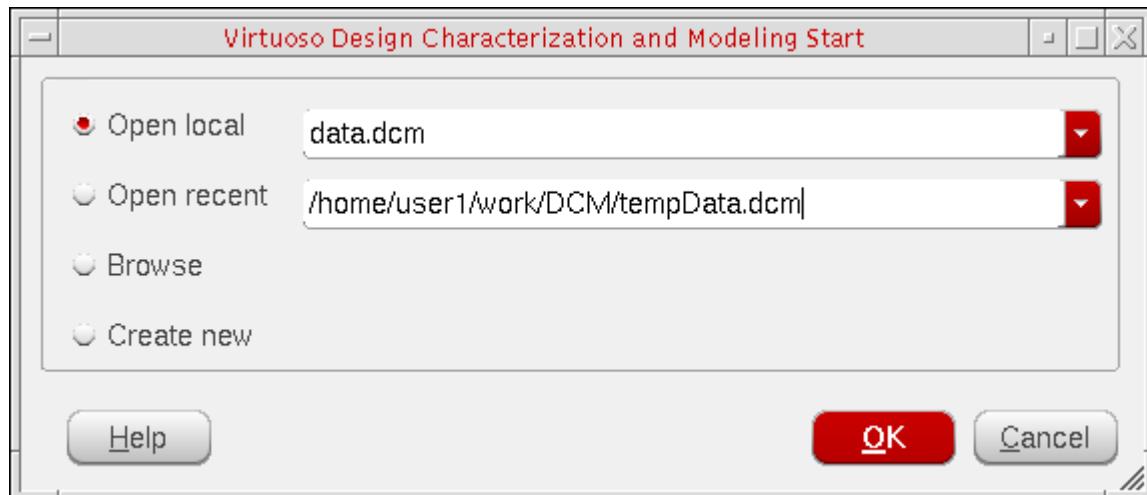
To begin verifying behavioral models, see Chapter 9, “Design Verification.”

See also dcmStartGUI for information that can help you get started with batch flow.

## Accessing the Start Form

You can access the Start form using either of the following methods:

- In the Command Interpreter Window, launch DCM.
  - In the Virtuoso Design Characterization and Modeling window, choose *File – Open*.
- The Virtuoso Design Characterization and Modeling Start form appears.



On this form, you can select one of the following tasks:

- Open local

# Virtuoso Analog Design Environment GXL User Guide

## Design Characterization and Modeling

---

- [Open recent](#)
- [Browse](#)
- [Create new](#)

## **Opening a Local DCM Data File**

To select a local DCM data file (.dcm) located in the directory where you started the software, do the following:

1. On the Start form, select *Open local*.
2. Select one of the available set of local data files from the drop-down list.
3. Click *OK*.

The information from the local data file appears on the tabs of the Virtuoso Design Characterization and Modeling window.

## **Opening a Recently-Opened DCM Data File**

To select a recently-opened (or recently-saved) DCM data file, do the following:

1. On the Start form, select *Open recent*.
2. Select a recently-opened data files from the drop-down list.
3. Click *OK*.

The information from the selected data file appears on the tabs of the Virtuoso Design Characterization and Modeling window.

## **Browsing for a DCM Data File to Load**

To open the Choose DCM Data File to Be Loaded form so that you can navigate to and select a DCM data file to load, do the following:

1. On the Start form, select *Browse*.
2. Click *OK*.  
The Choose DCM Data File to Be Loaded form appears.
3. Navigate to and select a data file.
4. Click *Select*.

The information from the selected data file appears on the tabs of the Virtuoso Design Characterization and Modeling window.

## **Creating a New DCM Data File**

To open the Virtuoso Design Characterization and Modeling window to the *Design* tab so that you can select a design, do the following:

1. On the Start form, select *Create new*.
2. Click *OK*.

The Virtuoso Design Characterization and Modeling window appears.

## Saving Changes



If an error prompt appears when you try to save changes, you must correct the errors before the program will save your changes. The software attempts to take you to the relevant tab to correct the error. See also “[Viewing the Last Error](#)” on page 211.

To save changes you make on any tab at any time, do the following:

- Click *Apply*.

The program saves updated information to the current DCM data file.

If this is the first time you are saving data, the Choose or Type a Data File Name form appears. You can navigate to and select an existing DCM data file to which to save your changes or you can type a new file name in the *File name* field on this form. Click *Save*.

Alternatively, you can save changes as follows:

- Choose *File – Save*.

The program saves updated information to the current DCM data file.

If this is the first time you are saving data, the Choose or Type a Data File Name form appears. You can navigate to and select an existing DCM data file to which to save your changes or you can type a new file name in the *File name* field on this form. Click *Save*.

To save your data to a different file name, do the following:

1. Choose *File – Save As*.

The Save As form appears.

2. Navigate to a directory location and select or type a file name in the *File name* field.

3. Click *Save*.

The program saves your data to the specified file name and directory location.

## Using the Tools Menu

You can use the *Tools* menu to do the following:



- *Run All* (same as [clicking the green Run button](#))
- *Run Verilog-A[MS]* performs Verilog-A[MS] model generation
- *Run Verilog-D* performs Verilog-D model generation
- *Run Liberty* performs Liberty model generation
- *Run Verification* performs design verification
- [\*Generate Liberty \(.lib\) library\*](#)
- [\*Start ADE XL\* opens \*dcmNameDCM\* in the \[ADE XL environment\]\(#\)](#)
- Note:** You can open the configuration (`configDCM`) or top cell view (`schematic`).
- [\*Edit Selected Function\* launches `dcmDeveloper` with the selected cell type](#)
- [\*Reload Selected Function\* reloads the current cell type to integrate any changes from `dcmDeveloper`](#)
- [\*Create New Function\* launches `dcmDeveloper` for new cell type creation](#)
- [\*Reload Function List\* reloads the function list on the Function tab to integrate any changes from `dcmDeveloper`](#)

## Viewing the Last Error

If you have errors in your DCM data and you try to save changes or exit the program, a prompt appears.



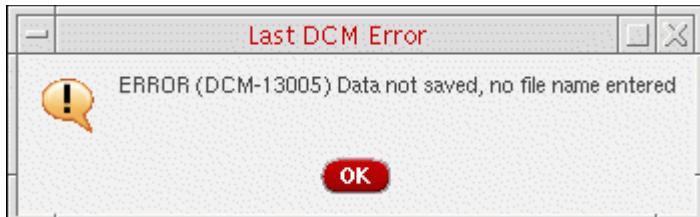
You must correct your errors before saving or exiting the program. The software attempts to take you to the relevant tab to correct the error.

To close the error prompt, do the following:

- Click *OK*.

To view the last error, do the following:

1. In the Virtuoso Design Characterization and Modeling window, choose *View – Last Error*.



A Last DCM Error prompt appears.

2. When you are finished viewing the last DCM error message, click *OK*.

The Last DCM Error prompt closes.

## Viewing Generated Files

You can view various files that DCM generates from the *View* menu in the Virtuoso Design Characterization and Modeling main window.



- [Viewing the Generation Log File](#) on page 214
- [Viewing Characterization and Modeling Details](#) on page 215
- [Viewing Any Text File](#) on page 217
- [Viewing Bottom-Up Verilog-A\[MS\] Generated Files](#) on page 218
- [Viewing Bottom-Up VHDL-AMS Generated Files](#) on page 220
- [Viewing Bottom-Up Verilog-D Generated Files](#) on page 222
- [Viewing Bottom-Up Liberty Generated Files](#) on page 223
- [Viewing Black Box Liberty \(.lib\) Files](#) on page 224
- [Viewing Verification Files](#) on page 225
- [Viewing Top-Down Verilog-A\[MS\] Generated Files](#) on page 227
- [Viewing Top-Down VHDL-AMS Generated Files](#) on page 228
- [Viewing the Results Database](#) on page 229

In general, while you are in the viewing window, you can highlight text and click *View* to open the following:

If your highlighted text contains...	the program opens...
■ a file name	the highlighted file in another viewing window.
■ a directory name	a <u>browser window</u> so that you can navigate into the highlighted directory.
■ a file name that ends with .rdb	the highlighted results file in a <u>results viewer window</u> .

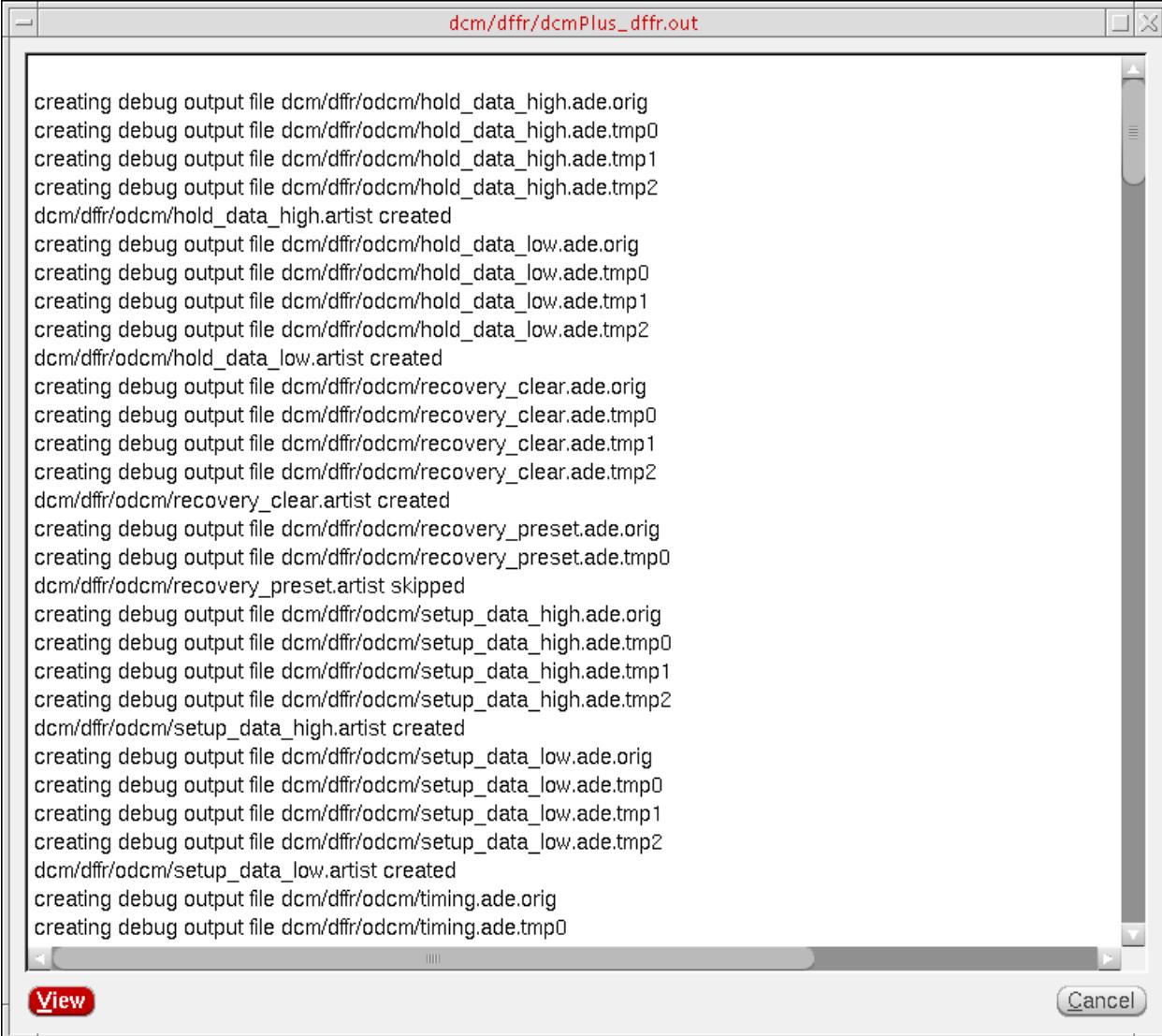
## Viewing the Generation Log File

To view the generation log file, do the following:

- Choose *View – Generation log*.

The generation log file appears in a viewing window.

The name and location of the file appears in the title banner.



The screenshot shows a window titled "dcm/dffr/dcmPlus\_dffr.out". The window contains a large amount of text representing a log of debug output files being created. The text is as follows:

```
creating debug output file dcm/dffr/odcm/hold_data_high.ad.e.orig
creating debug output file dcm/dffr/odcm/hold_data_high.ad.e.tmp0
creating debug output file dcm/dffr/odcm/hold_data_high.ad.e.tmp1
creating debug output file dcm/dffr/odcm/hold_data_high.ad.e.tmp2
dcm/dffr/odcm/hold_data_high.artist created
creating debug output file dcm/dffr/odcm/hold_data_low.ad.e.orig
creating debug output file dcm/dffr/odcm/hold_data_low.ad.e.tmp0
creating debug output file dcm/dffr/odcm/hold_data_low.ad.e.tmp1
creating debug output file dcm/dffr/odcm/hold_data_low.ad.e.tmp2
dcm/dffr/odcm/hold_data_low.artist created
creating debug output file dcm/dffr/odcm/recovery_clear.ad.e.orig
creating debug output file dcm/dffr/odcm/recovery_clear.ad.e.tmp0
creating debug output file dcm/dffr/odcm/recovery_clear.ad.e.tmp1
creating debug output file dcm/dffr/odcm/recovery_clear.ad.e.tmp2
dcm/dffr/odcm/recovery_clear.artist created
creating debug output file dcm/dffr/odcm/recovery_preset.ad.e.orig
creating debug output file dcm/dffr/odcm/recovery_preset.ad.e.tmp0
dcm/dffr/odcm/recovery_preset.artist skipped
creating debug output file dcm/dffr/odcm/setup_data_high.ad.e.orig
creating debug output file dcm/dffr/odcm/setup_data_high.ad.e.tmp0
creating debug output file dcm/dffr/odcm/setup_data_high.ad.e.tmp1
creating debug output file dcm/dffr/odcm/setup_data_high.ad.e.tmp2
dcm/dffr/odcm/setup_data_high.artist created
creating debug output file dcm/dffr/odcm/setup_data_low.ad.e.orig
creating debug output file dcm/dffr/odcm/setup_data_low.ad.e.tmp0
creating debug output file dcm/dffr/odcm/setup_data_low.ad.e.tmp1
creating debug output file dcm/dffr/odcm/setup_data_low.ad.e.tmp2
dcm/dffr/odcm/setup_data_low.artist created
creating debug output file dcm/dffr/odcm/timing.ad.e.orig
creating debug output file dcm/dffr/odcm/timing.ad.e.tmp0
```

At the bottom left of the window is a red "View" button, and at the bottom right is a "Cancel" button.

## Viewing Characterization and Modeling Details

To view characterization and modeling details, do the following:

- Choose *View – Documentation*.

The characterization and modeling output file appears in a viewing window.  
The name and location of the file appears in the title banner.

dcm/dffr/dcmGen/dffr.txt

dcm/dffr/dcmGen/dffr.txt: dffr documentation for dffr

Design:

```
Library: acvC25Lib
Cell: dffr
View: schematic
```

Q Output
QB Inverted Output
CLK Rising Clock
D Data
R Low Clear
gndd! Vss Low=0
vddd! Vdd High=3

Variable names and values:

```
dcm_temperature (Temperature) = 27 Degrees C List: 0 100
dcm_slope (Input slope) = 0.2n Seconds List: 0.1n 0.2n
dcm_global_vddd_value (Vdd voltage) = 3 Volts List: 90% 110%
dcm_load_cap (Output load) = 100f Farads List: 150f 250f 350f
```

Verilog-A model dcm/dffr/dcmGen/dffr.va:

```
Sweep dffr_Verilog_A_Timing measures Timing model method
Test dffr_timing
    R_Q_falling:t2 creates dffr_R_Q_falling_t2.vat
        calculated end time of Q rising edge (from R)
    R_QB_rising:t1 creates dffr_R_QB_rising_t1.vat
        calculated start time to QB rising edge (from R)
    R_Q_falling:delay creates dffr_R_Q_falling_delay.vat
        rising delay from R to Q
    CLK_Q_rising:t1 creates dffr_CLK_Q_rising_t1.vat
        calculated start time of Q rising edge (from CLK)
```

**View** **Cancel**

# Virtuoso Analog Design Environment GXL User Guide

## Design Characterization and Modeling

A list of SKILL files the program creates for setting up, characterizing, calibrating, and verifying design functions appears at the tail end of the file.

```
dcm/dffr/odcm/width_clk_low.artist/simulatorOptions > dcm/dffr/dcmGen/acvC25Lib/dffrDCM/spectre/dffr_width_clk_low.il  
dcm/dffr/odcm/width_clk_low.artist/spList > dcm/dffr/dcmGen/acvC25Lib/dffrDCM/spectre/dffr_width_clk_low.il  
dcm/dffr/odcm/width_clk_low.artist/variables > dcm/dffr/dcmGen/acvC25Lib/dffrDCM/spectre/dffr_width_clk_low.il  
Processing model file 'dff.va' of type Verilog-A  
From dcm/dffr/odcm/dff.va to dcm/dffr/dcmGen/dffr.va  
Processing model file 'top_down.va' of type Verilog-A  
Processing model file 'dff.va' of type Top-down Verilog-A  
Processing model file 'top_down.va' of type Top-down Verilog-A  
From dcm/dffr/odcm/top_down.va to dcm/dffr/dcmGen/dffr.tdva  
Processing model file 'dff.lib' of type Liberty  
From dcm/dffr/odcm/dff.lib to dcm/dffr/dcmGen/dffr.lib_gen  
Processing model file 'dff.v' of type Verilog-D  
From dcm/dffr/odcm/dff.v to dcm/dffr/dcmGen/dffr.v_gen  
Creating 'dcm/dffr/dcmGen/dffr_Verification_setup.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verification_characterize.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verification_verify.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_A_setup.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_A_characterize.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_A_calibrate.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_A_verify.il'  
Creating 'dcm/dffr/dcmGen/dffr_dffr.tdva.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_D_setup.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_D_characterize.il'  
Creating 'dcm/dffr/dcmGen/dffr_Verilog_D_calibrate.il'  
Creating 'dcm/dffr/dcmGen/dffr.Liberty_setup.il'  
Creating 'dcm/dffr/dcmGen/dffr.Liberty_characterize.il'  
Creating 'dcm/dffr/dcmGen/dffr.Liberty_calibrate.il'  
Creating 'dcm/dffr/dcmGen/dffr_setup.il'  
Creating 'dcm/dffr/dcmGen/dffr_setup_all.il'  
Creating 'dcm/dffr/dcmGen/dffr_load_procs.il'
```

## **Viewing Any Text File**

To view any text file, do the following:

1. Choose *View – File*.  
The Open File browser window appears.
2. Navigate to and select the file you want to view.
3. Click *Open*.  
The file you selected appears in a viewing window.
4. When you are done viewing the file, click *Cancel*.  
The viewing window closes.

## Viewing Bottom-Up Verilog-A[MS] Generated Files

You can view bottom-up Verilog-A[MS] files from the *View – Verilog-A[MS]* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

- Choose *View – Verilog-A[MS]* –

<i>Generated Model</i>	The generated Verilog-A[MS] model <i>dcm/dcmName/dcmGen/cellName.va</i> appears in a viewing window
<i>Calibrated Model</i>	The calibrated Verilog-A[MS] model <i>dcm/dcmName/dcmGen/cellName.va</i> appears in a viewing window
<i>Lookup Table</i>	A browser window appears so that you can navigate to and select a table file (*.vat) to view; when you click <i>Open</i> , the table file appears in a viewing window
<i>All Lookup Tables</i>	All calibrated look-up tables (.vat files) appear as a single file ( <i>dcm/dcmName/dcmPlus_cellName.tab</i> ) in a viewing window
<i>Setup Script</i>	The Verilog-A[MS] setup script <i>dcm/dcmName/dcmGen/cellName_Verilog_A_setup.il</i> appears in a viewing window
<i>Characterization Script</i>	The Verilog-A[MS] characterization script <i>dcm/dcmName/dcmGen/cellName_Verilog_A_characterize.il</i> appears in a viewing window
<i>Calibration Script</i>	The Verilog-A[MS] calibration script <i>dcm/dcmName/dcmGen/cellName_Verilog_A_calibrate.il</i> appears in a viewing window

## Virtuoso Analog Design Environment GXL User Guide

### Design Characterization and Modeling

---

- Choose *View – Verilog-A[MS]* –

<i>Verification Script</i>	The Verilog-A[MS] verification script <i>dcm/dcmName/dcmGen/cellName_Verilog_A_verify.il</i> appears in a viewing window
<i>Verification Results</i>	Verification results from the results database <i>dcmName_cellName_va_verify.rdb</i> appear in a <u>results database viewing window</u>
<i>HTML Verification Results</i>	Verification results appear in an HTML browser window ( <i>dcmName_cellName_va_verify.html</i> )
<i>Text Verification Results</i>	Verification results appear as comma-separated values in a viewing window ( <i>dcmName_cellName_va_verify.txt</i> )

The name and location of the file appears in the title banner.

## Viewing Bottom-Up VHDL-AMS Generated Files

You can view bottom-up VHDL-AMS files from the *View – Bottom-Up AMS* menu in the Virtuoso Design Characterization and Modeling window:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

► Choose *View – Bottom-Up AMS* –

<i>Generated Model</i>	The generated AMS models (Verilog-A[MS] and any other model formats that have been added as Top-Down AMS extras) appear, one in each viewing window
<i>Calibrated Model</i>	The calibrated AMS models (both Verilog-A[MS] and VHDL-AMS) appear, one in each viewing window
<i>Lookup Table</i>	A browser window appears so that you can navigate to and select a table file (*.vat) to view; when you click <i>Open</i> , the table file appears in a viewing window
<i>All Lookup Tables</i>	All calibrated look-up tables (.vat files) appear as a single file ( <i>dcm/dcmName/dcmPlus_cellName.tab</i> ) in a viewing window
<i>Setup Script</i>	The setup script <i>dcm/dcmName/dcmGen/cellName_AMS_setup.il</i> appears in a viewing window
<i>Characterization Script</i>	The characterization script <i>dcm/dcmName/dcmGen/cellName_AMS_characterize.il</i> appears in a viewing window
<i>Calibration Script</i>	The calibration script <i>dcm/dcmName/dcmGen/cellName_AMS_calibrate.il</i> appears in a viewing window

## Virtuoso Analog Design Environment GXL User Guide

### Design Characterization and Modeling

---

► Choose View – *Bottom-Up AMS* –

<i>Verification Script</i>	The verification script <code>dcm/dcmName/dcmGen/ cellName_AMS_verify.il</code> appears in a viewing window
<i>Verification Results</i>	Verification results from the results database <code>dcmName_cellName_va_verify.rdb</code> appear in a <u>results database viewing window</u>
<i>HTML Verification Results</i>	Verification results appear in an HTML browser window ( <code>dcmName_cellName_va_verify.html</code> )
<i>Text Verification Results</i>	Verification results appear as comma-separated values in a viewing window ( <code>dcmName_cellName_va_verify.txt</code> )

## Viewing Bottom-Up Verilog-D Generated Files

You can view bottom-up Verilog-D files from the *View – Verilog-D* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

- Choose *View – Verilog-D* –

<i>Generated Model</i>	The generated Verilog-D model <i>dcm/dcmName/dcmGen/cellName.v_gen</i> appears in a viewing window
<i>Calibrated Model</i>	The calibrated Verilog-D model <i>dcm/dcmName/dcmGen/cellName.v</i> appears in a viewing window
<i>User Defined Primitive</i>	User-defined primitives in <i>dcm/dcmName/dcmGen/cellName_prim.v</i> appear in a viewing window, if the file exists
<i>Setup Script</i>	The Verilog-D setup script <i>dcm/dcmName/dcmGen/cellName_Verilog_D_setup.il</i> appears in a viewing window
<i>Characterization Script</i>	The Verilog-D characterization script <i>dcm/dcmName/dcmGen/cellName_Verilog_D_characterize.il</i> appears in a viewing window
<i>Calibration Script</i>	The Verilog-D calibration script <i>dcm/dcmName/dcmGen/cellName_Verilog_D_calibrate.il</i> appears in a viewing window

The name and location of the file appears in the title banner.

## Viewing Bottom-Up Liberty Generated Files

You can view bottom-up Liberty files from the *View – Liberty* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

► Choose *View – Liberty* –

<i>Generated Model</i>	The generated Liberty model <i>dcm/dcmName/dcmGen/cellName.lib_gen</i> appears in a viewing window
<i>Calibrated Mode</i>	The calibrated Liberty model <i>dcm/dcmName/dcmGen/cellName.lib</i> appears in a viewing window
<i>Generated Library</i>	The generated Liberty library <i>outputLibraryFileName.lib</i> appears in a viewing window
<i>Setup Script</i>	The Liberty setup script <i>dcm/dcmName/dcmGen/cellName_Liberty_setup.il</i> appears in a viewing window
<i>Characterization Script</i>	The Liberty characterization script <i>dcm/dcmName/dcmGen/cellName_Liberty_characterize.il</i> appears in a viewing window
<i>Calibration Script</i>	The Liberty calibration script <i>dcm/dcmName/dcmGen/cellName_Liberty_calibrate.il</i> appears in a viewing window

The name and location of the file appears in the title banner.

## Viewing Black Box Liberty (.lib) Files

You can view Black Box Liberty (.lib) files from the *View – Black Box Liberty (.lib)* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

- Choose *View – Black Box Liberty (.lib)* –

<i>Generated Model</i>	The generated Black Box Liberty model <i>dcm/dcmName/dcmGen/cellName.lib_gen</i> appears in a viewing window
<i>Calibrated Model</i>	The calibrated Black Box Liberty model <i>dcm/dcmName/dcmGen/cellName.lib</i> appears in a viewing window
<i>Calibration Script</i>	The Black Box Liberty calibration script <i>dcm/dcmName/dcmGen/cellName.Liberty_calibrate.il</i> appears in a viewing window
<i>Generated Library</i>	The generated Black Box library <i>outputLibraryFileName.lib</i> appears in a viewing window

## Viewing Verification Files

You can view verification results from the *View – Verification* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

- Choose *View – Verification* –

<i>Setup Script</i>	The verification setup script <i>dcm/dcmName/dcmGen/</i> <i>cellName_Verification_setup.il</i> appears in a viewing window
<i>Characterization Script</i>	The verification characterization script <i>dcm/dcmName/dcmGen/</i> <i>cellName_Verification_characterize.il</i> appears in a viewing window
<i>Verification Script</i>	The verification script <i>dcm/dcmName/dcmGen/</i> <i>cellName_Verification_verify.il</i> appears in a viewing window
<i>Simulation Results</i>	Simulation results from the results database <i>dcmName_design_verification_characterize.rdb</i> appear in a <u>results database viewing window</u>
<i>Verification Results</i>	Verification results from the results database <i>dcmName_Verification_verify.rdb</i> appear in a <u>results database viewing window</u>
<i>HTML Verification Results</i>	Verification results appear in an HTML browser window ( <i>dcmName_Verification_verify.htm</i> l)
<i>Text Verification Results</i>	Verification results appear as comma-separated values in a viewing window ( <i>dcmName_Verification_verify.txt</i> )

**Virtuoso Analog Design Environment GXL User Guide**  
Design Characterization and Modeling

---

The name and location of the file appears in the title banner.

## **Viewing Top-Down Verilog-A[MS] Generated Files**

You can view top-down generated files from the *View – Top Down Verilog-A[MS]* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

- Choose *View – Top Down Verilog-A[MS]* –

<i>Generated Model</i>	The generated top-down model <i>dcm/dcmName/dcmGen/cellName.tdva</i> appears in a viewing window
<i>Script</i>	The top-down script <i>dcm/dcmName/dcmGen/dcmName_cellName.tdva.il</i> appears in a viewing window

The name and location of the file appears in the title banner.

## **Viewing Top-Down VHDL-AMS Generated Files**

You can view top-down generated files from the *View – Top Down AMS* menu in the Virtuoso Design Characterization and Modeling window as follows:

**Note:** Where *dcmName* appears in the following table, it stands for the prefix of the DCM data file name you typed when you first saved your DCM data (*dcmName.dcm*). See “[Saving Changes](#)” on page 209 for more information. *cellName* stands for the name that appears in the *Cell Name* field on the *Design* tab.

- Choose *View – Top Down AMS* –

<i>Generated Model</i>	The generated top-down AMS models (Verilog-A[MS] and any other model formats that have been added as Top-Down AMS extras) appear, one in each viewing window
<i>Script</i>	The top-down script <code>dcm/dcmName/dcmGen/dcmName_cellName.tdva.il</code> appears in a viewing window

The name and location of the file appears in the title banner.

## **Viewing the Results Database**

You can view individual results from individual runs by looking in the results database.

To view the results database, do the following:

- Choose *View – Result Database*.

# Virtuoso Analog Design Environment GXL User Guide

## Design Characterization and Modeling

The results database viewing window appears.

The screenshot shows a software interface titled "Virtuoso Analog Design Environment GXL User Guide". The main window is titled "/home/jillw/work/DCM/data\_Verilog\_A\_Timing\_characterize.rdb". The interface includes a toolbar with icons for file operations like Open, Save, Print, and Search. Below the toolbar is a menu bar with "File" and the Cadence logo. The main area displays a results database with three tables:

	1	2	3
status	done	done	done

	1	2	3
dcm_global_vddd_value	2.7	3	3.3

	1	2	3
up_clk_tran:t2	1.071n	993.2p	937.4p
up_clk_tran:end_val	1.349n	1.438n	1.565n
dn_ref_tran:delay	1.028n	948.8p	897.9p
up_ref_tran:t1	474.7p	438.9p	409.6p
up_clk_tran:slope	183p	174p	146.1p
dnb_clk_tran:slope	492.5p	461.9p	417.7p
dnb_clk_tran:t1	214.4p	198.4p	222.5p
dnb_ref_tran:end_val	2.7	3	3.3
up_clk_tran:delay	987p	914.1p	867.5p
upb_ref_tran:delay	433.8p	403.6p	399.6p
dn_ref_tran:t2	1.113n	1.029n	968.4p
dnb_clk_tran:delay	424p	394.4p	390.7p
dnb_ref_tran:t2	1.128n	1.043n	938.2p
dnb_ref_tran:t1	679.7p	626.9p	579p
dnb_ref_tran:delay	919.2p	849.1p	784p
up_clk_tran:t1	888.4p	819.2p	791.2p
dn_ref_tran:slope	184.7p	175.9p	147.8p
upb_ref_tran:slope	497.5p	466.2p	420.5p
upb_ref_tran:t1	221p	204.6p	229.3p
dn_clk_tran:delay	593p	549.7p	509.7p
dnb_clk_tran:end_val	-28.18n	-25.07n	-21.59n
upb_clk_tran:slope	443.4n	411.5n	356.2n

## Exiting Design Characterization and Modeling

To exit the Virtuoso Design Characterization and Modeling window, do the following:

- Choose *File – Exit*.

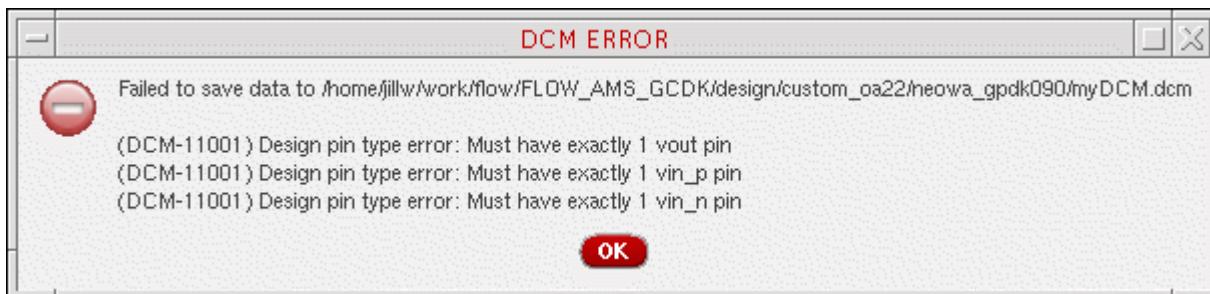
If you have not saved your data, a prompt appears.

- Click *Yes* to save.

If this is the first time you are saving data, the Choose or Type a Data File Name form appears. You can navigate to and select an existing DCM data file to which to save your changes or you can type a new file name in the *File name* field on this form. Click *Save*.

- Click *No* to choose not to save.

If you have errors in your DCM data, a prompt appears.



You must correct your errors before exiting. The software attempts to take you to the relevant tab to correct the error. See also [“Viewing the Last Error”](#) on page 211.

**Virtuoso Analog Design Environment GXL User Guide**  
Design Characterization and Modeling

---

---

# Model Generation

---

You can create silicon-calibrated models in views from original schematic designs using the Design Characterization and Modeling (DCM) feature of the Virtuoso® Analog Design Environment GXL (ADE GXL).

See the following topics for more information:

- [Specifying a Design](#) on page 234  
(for bottom-up modeling, design verification, and Liberty MS model generation, begin here)
- [Specifying a Function](#) on page 241  
(for top-down modeling, begin here)
- [Specifying Modeling Requirements](#)
  - [Specifying Top-Down Modeling for AMS Models](#) on page 253
  - [Specifying Bottom-Up Modeling for AMS Models](#) on page 259
  - [Specifying Bottom-Up Modeling for Verilog-D](#) on page 263
  - [Specifying Bottom-Up Modeling for Liberty](#) on page 265
- [Setting Up Model Calibration Requirements](#) on page 268
  - [Calibrating Models](#) on page 269
  - [Modifying Calibration Sweep Parameters](#) on page 271
  - [Specifying Parameters](#) on page 278
  - [Specifying Options](#) on page 281
- [Generating Models and Running Tests](#) on page 282
- [Generating Liberty MS Models](#) on page 287

## Specifying a Design

You specify design information for bottom-up modeling, design verification, and Liberty .lib model generation on the *Design* tab. You can select, browse for, or type your design information. See the following topics for details:

- [Selecting an Instance on the Schematic](#) on page 234
- [Selecting a Library, Cell, and View](#) on page 235
- [Typing the Library, Cell, View, and Pin Information](#) on page 235

See also [“Enabling Design Verification”](#) on page 317.

**Note:** For top-down modeling, you start by specifying a function on the [Function](#) tab.

See the following sections for more information:

- [Opening a Design from the Design Tab](#) on page 236
- [Selecting the Testbench Library](#) on page 236
- [Opening the Testbench Schematic](#) on page 236
- [Loading Model Setup and Simulator Options from an ADE State Directory](#) on page 237

### Selecting an Instance on the Schematic

With your schematic open, do the following:

1. On the *Design* tab of the Virtuoso Design Characterization and Modeling window, click *Select*.

Your already-open design schematic appears into the foreground so that you can select a cell instance from the schematic.

2. In the schematic window, select a cell instance.

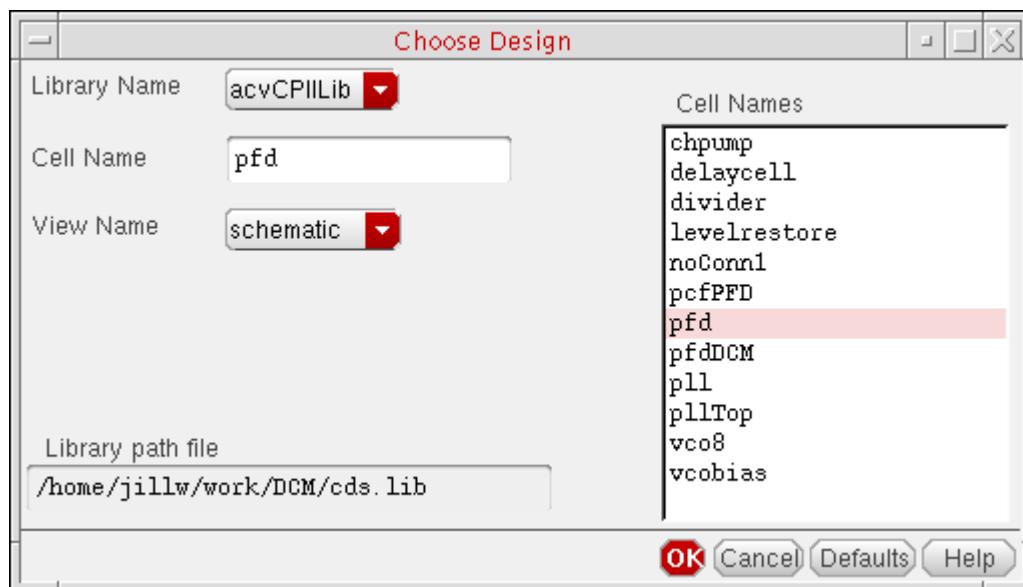
The library name, cell name, view name, and pins appear in their respective fields on the *Design* tab of the Virtuoso Design Characterization and Modeling window.

## Selecting a Library, Cell, and View

To select a library, cell, and view, do the following:

1. On the *Design* tab of the Virtuoso Design Characterization and Modeling window, click *Browse*.

The Choose Design form appears.



2. In the *Library Name* drop-down list, select a library.
3. In the *Cell Names* scrolling list, select a cell.
4. In the *View Name* drop-down list, select a view.
5. Click *OK*.

The library name, cell name, view name, and pins appear in their respective fields on the *Design* tab in the Virtuoso Design Characterization and Modeling window.

## Typing the Library, Cell, View, and Pin Information

To specify the design by typing the library, cell, view, and pin information in the fields on the *Design* tab, do the following:

1. In the *Library Name* drop-down list, select a library.
2. In the *Cell Name* field, type a valid cell name.

3. In the *View Name* field, type a valid view name.
4. In the *Pins* field, type a space-separated list of the pin names.



The *Select* and *Browse* methods are required for designs that have inherited connections or globals because they return additional information from the schematic editor about these entities used in the design.

## Opening a Design from the Design Tab

To open the design in a schematic window, do the following:

- Click *Open Design*.

The design specified in the fields on the *Design* tab appears in a schematic window.

## Selecting the Testbench Library

To specify the library where you want to put the testbench schematic (for bottom-up design or design verification), do the following:

- In the *Testbench Library* drop-down list, select a library.

## Opening the Testbench Schematic

DCM generates a testbench schematic when you click *Generate*. The cell name of the testbench schematic (for bottom-up design or design verification) is *cellNameDCM*, where *cellName* is what appears in the *Cell Name* field on the *Design* tab. The library is whatever you selected in the *Testbench Library* drop-down list.

To open the testbench schematic, do the following:

- Click *Open Testbench*.

The testbench schematic appears in the schematic window.

**Note:** If, instead, you get the following message, a testbench schematic does not exist.

Most likely, you have not yet clicked Generate.



## **Loading Model Setup and Simulator Options from an ADE State Directory**

You can load an ADE state or directory containing process information, model libraries, and simulation options for DCM to merge with the test it creates.

To load model setup and simulator options, do the following:

1. In the *ADE State Containing Model Setup and Simulator Options* group box, click ....

The State Browser to use form appears.



## **Loading ADE State**

To load the data from an ADE state:

1. Click *ADE* to load an ADE state.

## **Virtuoso Analog Design Environment GXL User Guide**

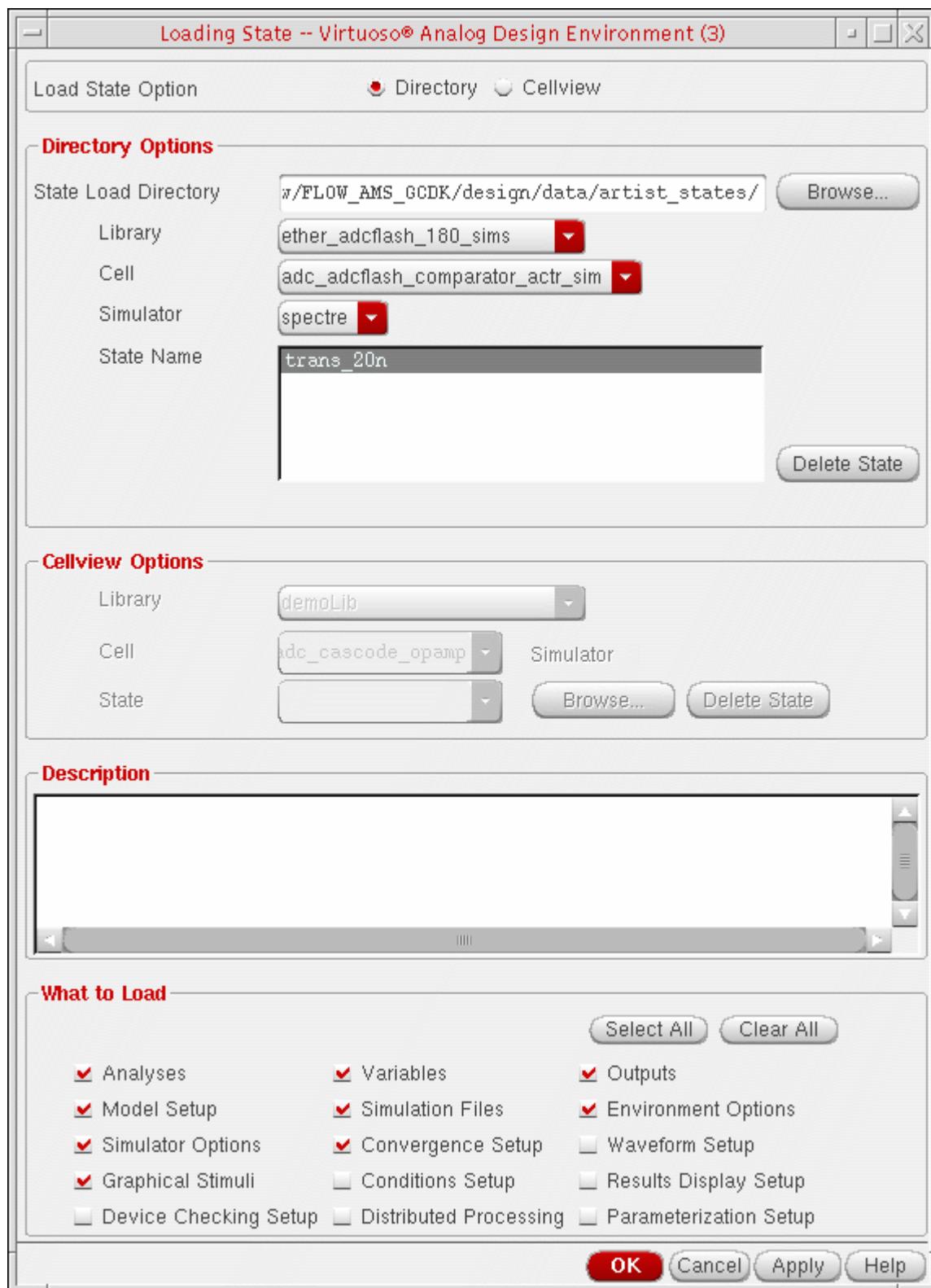
### Model Generation

---

The Loading State form appears.

# Virtuoso Analog Design Environment GXL User Guide

## Model Generation



2. Select an ADE state.
3. Click *OK*.

The state you selected appears in the *State Directory* field on the *Design* tab.

## **Loading from a Directory**

To load the data from a directory:

1. Click *Directory*.  
The Select State Directory form appears.
2. Navigate to the directory you want to load.
3. Click *Open*.

## Specifying a Function

The *Function* tab is where you describe design functionality (cell type, pin types, voltage ranges). To select and load a design function and specify pin types, do the following:

1. In the *Function* list area, select a design function.
2. Click *Load Selected Function* (or double-click your selection).

Pin types for the selected design function appear in the *Design pin types* list area.

Design pin types		
	Pin Type	Attribute
dn	Inverted Down Output	
dnb	Inverted Down Output	
up	Up Output	
upb	Inverted Up Output	
clk	Rising VCO Clock	
ref	Rising Ref Clock	
reset	High Reset	
vddd	Rising VCO Clock	
gndd	Vss	

3. (Optional) Enable one or both of the following advanced pin options:
  - Differential*
  - Note:** This feature is not available for all cell types.
  - Components*
4. For each design pin listed in the first column, use the drop-down list in the *Pin Type* column to specify a pin type.

### *Important*

For top-down modeling, see also “[Specifying Design Pin Types for Top-Down Modeling](#)” on page 242.

5. Specify design pin attributes.

See the following topics for more information:

- [Selecting a Simulator](#) on page 243
- [Filtering Design Functions](#) on page 243

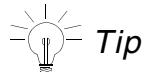
- [Viewing Design Function Data](#) on page 247
- [Enabling Differential Input/Output Pin Types](#) on page 250
- [Specifying Additional Components between a Pin and Ground](#) on page 250
- [Specifying Design Pin Attributes](#) on page 251
- [Viewing Function Help](#) on page 252

## Specifying Design Pin Types for Top-Down Modeling

When using Virtuoso® Design Characterization and Modeling (DCM) for top-down modeling, you can change pin names, add or remove pins, and move pins up and down in the *Design pin types* table.

To change a pin name, do the following:

- In the *Pin Name* column, click a pin name and type a new name.



You can also double-click a pin name to make the edit cursor appear so that you can edit a pin name.

To add or remove pins or move them up or down in the table, do the following:

1. In the *Design pin types* table, right-click the row for a pin.

A pop-up menu appears.

- Add Pin
- Remove Pin
- Move Pin Up
- Move Pin Down

2. Select the action you want to take from the pop-up menu:

*Add Pin*

The program adds a pin row after the selected pin row in the table and assigns an initial pin name that is the selected pin's name with a *1* suffix. The pin type is that same as the pin type of the selected pin.

*Remove Pin*

The program removes the selected pin row from the table.

*Move Pin Up*      The program moves the pin row up one position in the table.

*Move Pin Down*      The program moves the pin row down one position in the table.

## Selecting a Simulator

To select a simulator, do the following:

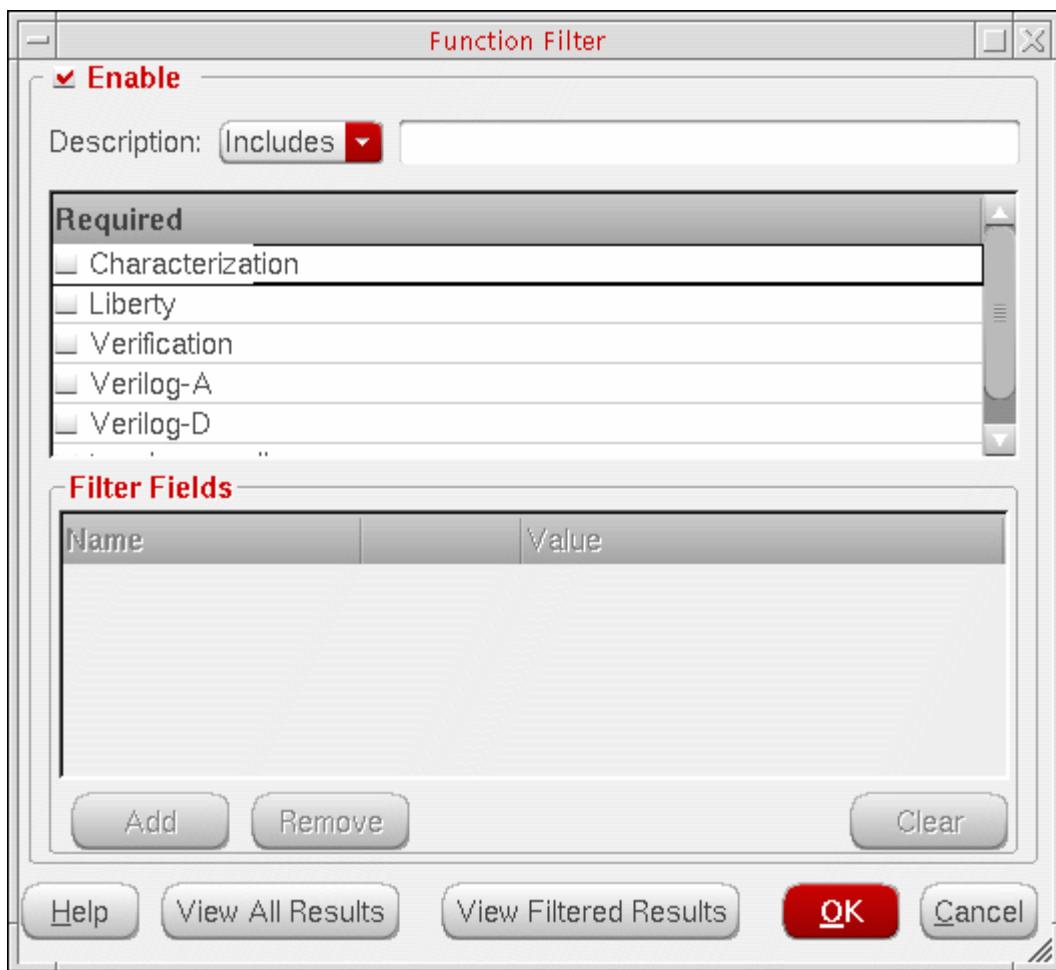
- In the *Simulator* drop-down list on the *Function* tab, select a simulator.

## Filtering Design Functions

To filter the design functions that appear in the *Function* list area on the *Function* tab, do the following:

1. On the *Function* tab, click *Filter List*.

The Function Filter form appears.



A mark appears in the *Enable* check box at the top left corner.

**Note:** To disable filtering, remove the mark from the *Enable* check box.

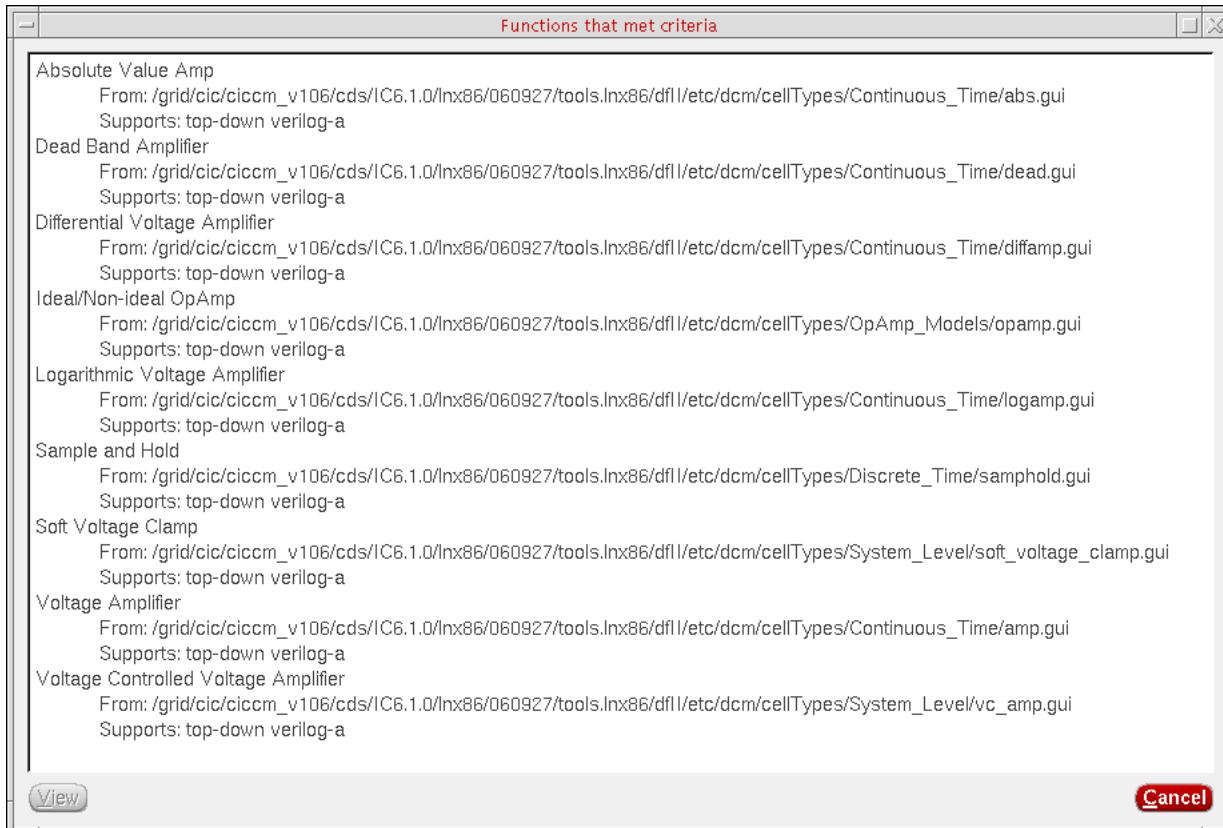
2. (Optional) In the *Description* drop-down list, choose one of the following:

<i>Ignore</i>	Does not use description text as a means of filtering design functions
<i>Includes</i>	Finds design functions whose descriptions contain the string you type in the field
<i>Excludes</i>	Finds design functions whose descriptions do not contain the string you type in the field
<i>Exactly</i>	Finds design functions whose descriptions exactly match the string you type in the field; the entire description text must match

# Virtuoso Analog Design Environment GXL User Guide

## Model Generation

**Note:** To view the set of design functions that meet the filter criteria, click *View Filtered Results*.



Functions that meet the filter criteria, the location of their .gui files, and what each function supports appears in a viewing window. When you are finished viewing the filtered results, click *Cancel* to close the window.

3. (Optional) To filter design functions by what they support, mark the check box of each item or items you want your selected design function to support:

<i>Characterization</i>	Supports model characterization
<i>Liberty</i>	Supports bottom-up Liberty behavioral model generation
<i>Verilog-A</i>	Supports bottom-up Verilog-A behavioral model generation
<i>Verilog-D</i>	Supports bottom-up Verilog-D behavioral model generation
<i>top-down verilog-a</i>	Supports top-down Verilog-A behavioral model generation
<i>Verification</i>	Supports design verification

**Note:** Again, you can view the set of design functions that meet the filter criteria by clicking *View Filtered Results*. When you are finished viewing the filtered results, click *Cancel* to close the window.

**4. Click *OK*.**

The filtered set of design functions appears in the *Function* list area.

See also [“Viewing Design Function Data”](#) on page 247.

## **Viewing Design Function Data**

You can view a list of all design functions, the location and name of the .gui file that goes with each function, and the set of DCM design methodologies that each function supports. You can also view the content of the .gui file.

To view design function data, do the following:

1. On the *Function* tab, click *Filter List*.

The Function Filter form appears.

2. On the Function Filter form, click *View All Results*.

## Virtuoso Analog Design Environment GXL User Guide

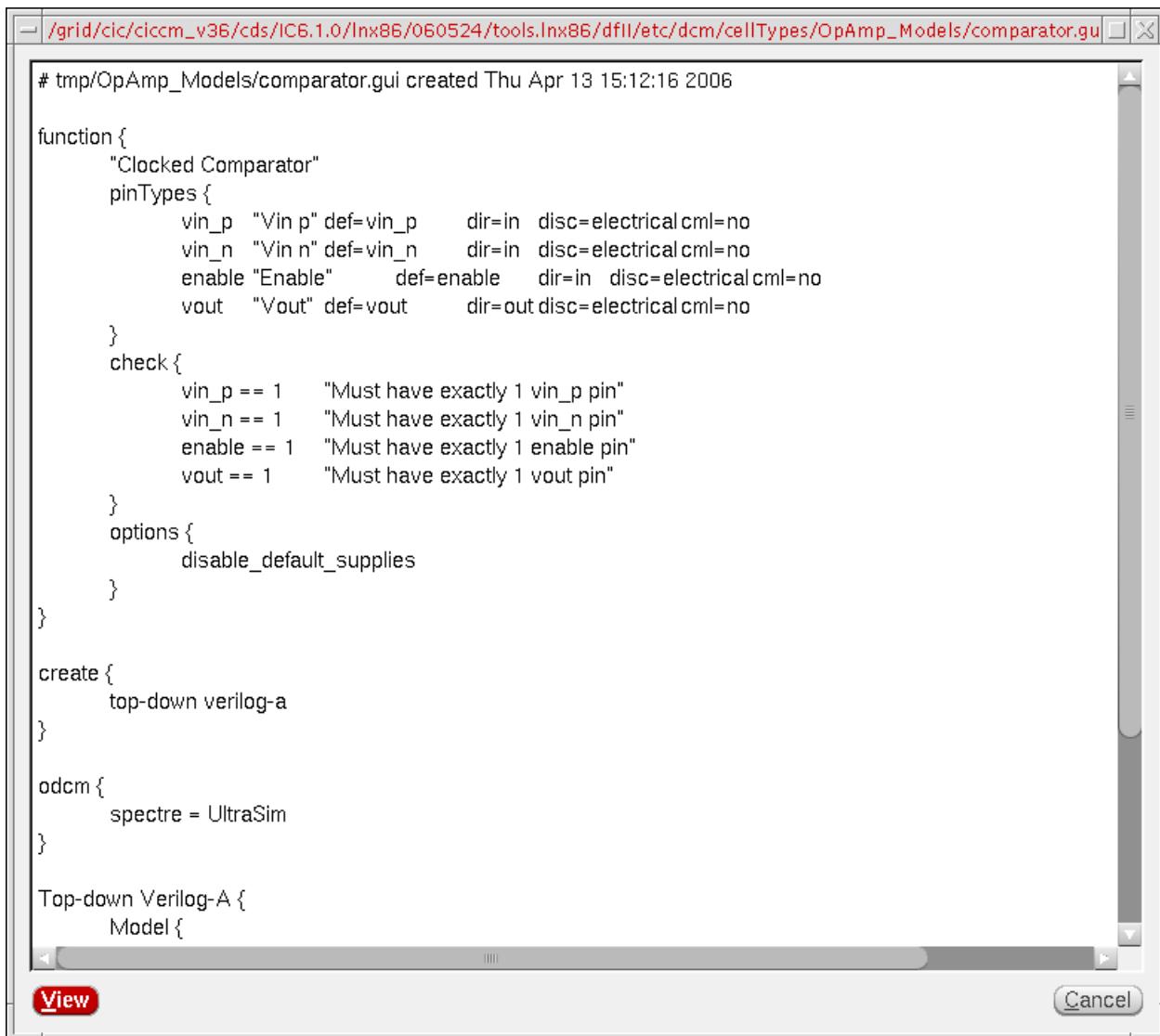
### Model Generation

The design functions appear in a Function Data viewing window. If the information is available, you will see the location and name of each `.gui` file that goes with each design function and a list of which DCM design methodologies each one supports.



**3.** (Optional) To view the content of the .gui file, highlight the *From* line and click *View*.

The content of the .gui file appears in a window.



```
/grid/cic/ciccm_v36/cds/IC6.1.0/lnx86/060524/tools.lnx86/dfl/etc/dcm/cellTypes/OpAmp_Models/comparator.gui

# tmp/OpAmp_Models/comparator.gui created Thu Apr 13 15:12:16 2006

function {
    "Clocked Comparator"
    pinTypes {
        vin_p "Vin p" def=vin_p      dir=in disc=electrical cml=no
        vin_n "Vin n" def=vin_n      dir=in disc=electrical cml=no
        enable "Enable"   def=enable   dir=in disc=electrical cml=no
        vout   "Vout"    def=vout     dir=out disc=electrical cml=no
    }
    check {
        vin_p == 1    "Must have exactly 1 vin_p pin"
        vin_n == 1    "Must have exactly 1 vin_n pin"
        enable == 1   "Must have exactly 1 enable pin"
        vout == 1     "Must have exactly 1 vout pin"
    }
    options {
        disable_default_supplies
    }
}

create {
    top-down verilog-a
}

odcm {
    spectre = UltraSim
}

Top-down Verilog-A {
    Model {
```

When you are finished viewing the file, click *Cancel* to close the window and return to the [Function Data viewing window](#).

**4.** When you are finished viewing the information in the Function Data viewing window, click *Cancel* to close the window.

## Enabling Differential Input/Output Pin Types

**Note:** This feature is not available for all cell types.

You can use differential pins to define one pin as having an inverted relationship to another pin (for example, a pin *cp* that is a rising-edge clock pin and *cpn* defined as *!cp*). A differential pin inherits its low and high voltages from the related pin. You can use this option for Current-Mode Logic (CML) designs.

To enable differential input/output pin types, do the following:

- On the *Function* tab, mark the *Differential* check box.

Differential pins appear in the drop-down list for each pin type.

See “[Specifying a Function](#)” on page 241 for information about selecting pin types.

Differential pins are indicated by a *!* prefix in the *Pin Type* drop-down list—for example, the differential pin for *clkin* appears as *!clkin*.

## Specifying Additional Components between a Pin and Ground

To specify additional components in your testbench between a pin of your design function and ground, do the following:

1. On the *Function* tab, mark the *Components* check box.

The following additional items appear on the *Pin Type* cyclic menu for each pin:

- Capacitor*
- Resistor*
- Voltage* (source)
- Current* (source)

2. In the drop-down list in the *Pin Type* column, select a component.

3. Click *Attributes* to [specify a value for the selected component](#).

4. On the Design Pin Attributes form, click *OK*.

The value appears in the *Attribute* column in the *Design pin types* group box.

## Specifying Design Pin Attributes

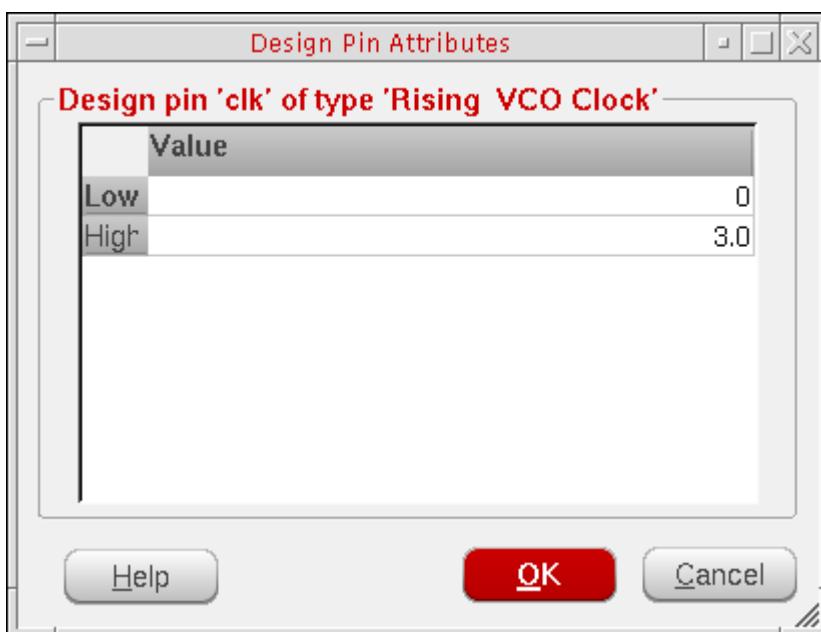
You can use design pin attributes to specify different attributes of a design function such as low and high voltage values for a design pin, a value for a primitive component between the pin and ground, or count for a divider.

To specify design pin attributes, do the following:

1. Double-click in the *Attribute* column for the pin.

Alternatively, you can select a pin and click *Attributes*.

The Design Pin Attributes form appears.



2. In each field that appears on the form, type the value you want.

For a design pin, type low and high voltage values.

For a component between the pin and ground, type the component's value.

3. Click *OK*.

The value or values you specified appear in the *Attribute* column in the *Design pin types* group box.

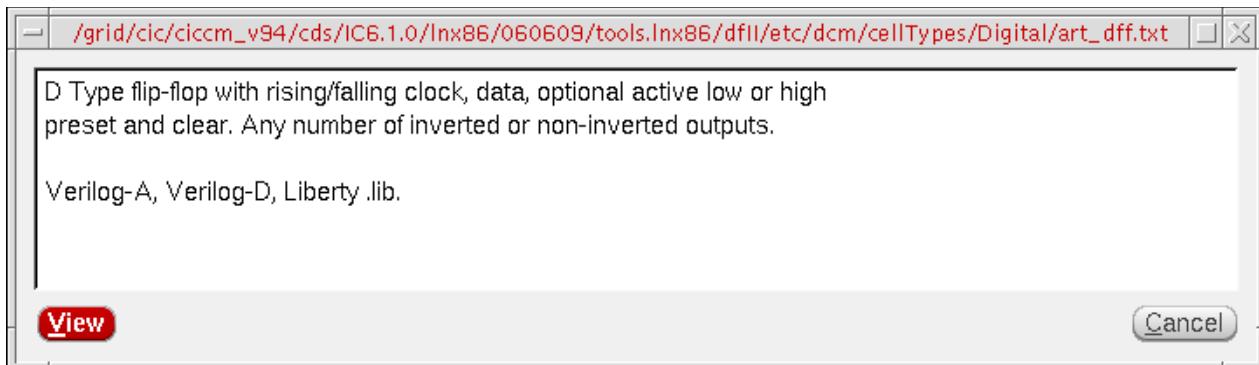
See also [“Pin Field Label”](#) on page 438 for information about row labels that appear on this form.

## **Viewing Function Help**

To view help text for the function you selected, do the following:

- Choose *View – Function Help*.

Function help text, if it exists, appears in a viewing window.



**Note:** The program looks for function help text files in `your_install_dir/dfII/etc/dcm/cellTypes`. The design function categories that appear in the *Select design function* tree in the *Function* list area on the *Function* tab determine the directory the program searches. For example, the program searches the Digital directory for *Dff* function help.

## Specifying Modeling Requirements

This section describes how to specify modeling requirements, including:

- [Specifying Top-Down Modeling for AMS Models](#), below
- [Specifying Bottom-Up Modeling for AMS Models](#) on page 259
- [Specifying Bottom-Up Modeling for Verilog-D](#) on page 263
- [Specifying Bottom-Up Modeling for Liberty](#) on page 265

Once you have specified modeling requirements, you can move on to [Setting Up Model Calibration Requirements](#).

### Specifying Top-Down Modeling for AMS Models

Once you have [selected a function](#), you can begin top-down modeling of a Verilog-A[MS], VHDL-AMS, and any other AMS model formats.

To begin:

- Select the *Top-Down* tab to view the Top-Down page.

You can now do the following:

- [Specifying Top-Down Modeling for Verilog-A\[MS\]](#)
- [Specifying Top-Down Modeling for VHDL-AMS](#)
- [Specifying Top-Down Advanced Verilog-A Options](#)

#### Specifying Top-Down Modeling for Verilog-A[MS]

1. Select the *Verilog-A[MS]* tab.
2. Select the *Create* check box.

**Note:** The *Create* check box is in the upper right corner of the tab.

The model parameters and options relevant for top-down design of [the function you selected](#) appear on the tab.



#### *Important*

What parameters appear in the *Parameter values* table depends on the design pins, their types, the selected cell type, and sometimes what options the model specifies. To refresh the *Parameter values* list if you make changes to any of this information, click *Get parameters*.

**Note:** As part of the process of creating the parameter list, the software verifies and saves the current DCM setup data to a temporary name. DCM does not let you save incorrect setup data; therefore, you must correct any setup data errors before you can enable the model type.

3. (Optional) Click *Destination* to open the [Model Destination form](#) for specifying a library destination for the behavioral model the program generates. Note that this form is specific to Verilog-A[MS].

See the following topics for more information:

- [Specifying Top-Down AMS Model Options](#) on page 255
- [Specifying Top-Down AMS Parameter Values](#) on page 255
- [Specifying Top-Down Advanced Verilog-A Options](#) on page 256

**Note:** Which items are available depends on which cell type you select.

### Specifying Top-Down Modeling for VHDL-AMS

Once you have [selected a function](#), you can begin top-down modeling of a VHDL-AMS models.



#### *Important*

You must set the `AXL_DCM_GUI_EXTRAS` environment variable to utilize the VHDL-AMS flow.

1. Select the *AMS* tab.
2. Select the *Create* check box.

**Note:** The *Create* check box is in the upper right corner of the tab.

The model parameters and options relevant for top-down design of [the function you selected](#) appear on the tab.



The parameters that appear in the *Parameter values* table depends on the design pins, their types, the selected cell type, and sometimes what options the model specifies. To refresh the *Parameter values* list if you make changes to any of this information, click *Get parameters*.

**Note:** As part of the process of creating the parameter list, the software verifies and saves the current DCM setup data to a temporary name. DCM does not let you save incorrect setup data; therefore, you must correct any setup data errors before you can enable the model type.

See the following topics for more information:

- [Specifying Top-Down AMS Model Options](#) on page 255
- [Specifying Top-Down AMS Parameter Values](#) on page 255
- [Specifying Top-Down Advanced Verilog-A Options](#) on page 256

**Note:** Which items are available depends on which cell type you select.

### **Specifying Top-Down AMS Model Options**

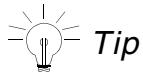
To specify model options for top-down AMS model generation, do the following:

1. In the *Options* group box, mark the check box for each option you want to enable.
2. Click in the *Value* column to edit any option value you want to change.

### **Specifying Top-Down AMS Parameter Values**

To add or change a parameter value, do the following:

1. Click in the *Value* column.
2. Type a parameter value.



To specify the results from a run, right-click on the parameter and choose *Get Measured Result*, then [choose the results from an ADE XL run](#)

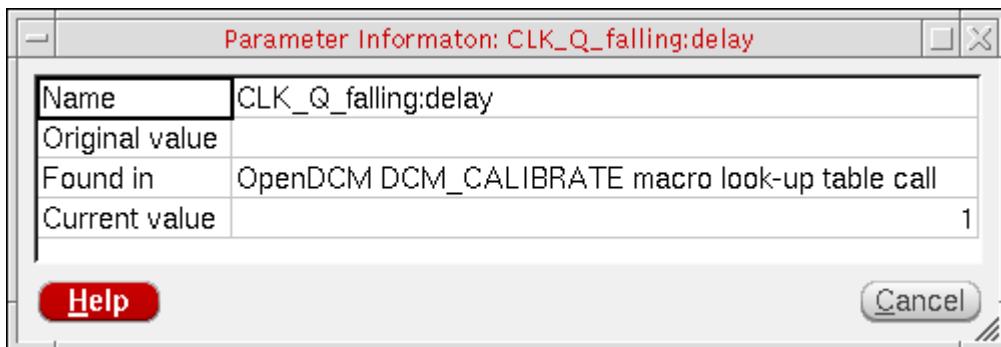
3. (Optional) Press *Tab* or *Return*.

The program uses the value you type where the look-up table call would be if this were a bottom-up model.

(Optional) To view detailed information about a parameter, do the following:

1. In the *Parameter values* group box on the *Top-Down Verilog-A[MS]* tab, double-click a parameter row.

A Parameter Information window appears. This window is for viewing information only.



The parameter name appears in the *Name* field.

The original parameter value appears in the *Original value* field.

Where the program found the parameter appears in the *Found in* field.

The current parameter value appears in the *Current value* field.

**Note:** You can read about the OpenDCM DCM\_CALIBRATE function, which appears on the sample form shown above, in [Appendix A, “OpenDCM.”](#)

2. When you are finished viewing information, click *Cancel*.

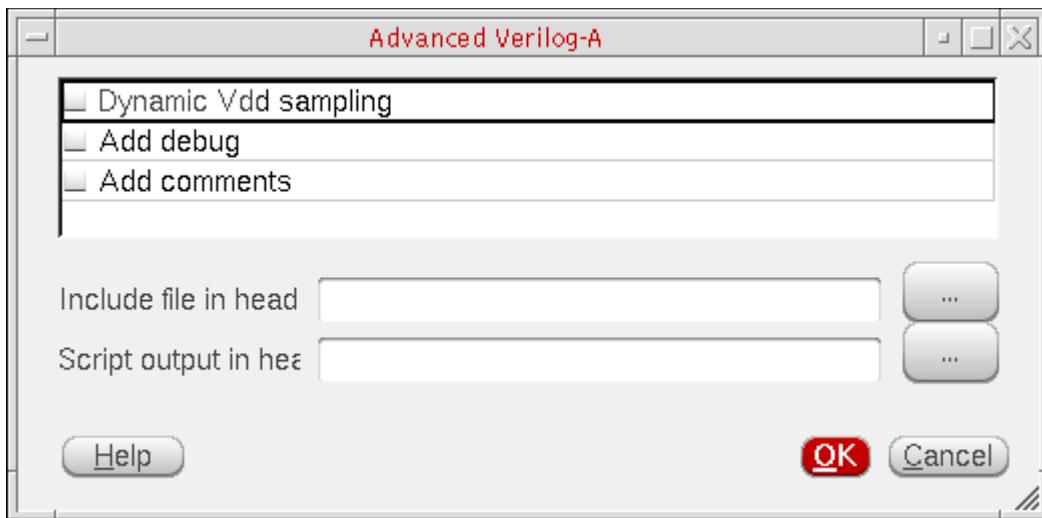
The Parameter Information window closes.

## Specifying Top-Down Advanced Verilog-A Options

To specify advanced Verilog-A options for top-down modeling, do the following:

1. On the *Top-Down Verilog-A[MS]* tab, click *Advanced*.

The Advanced Verilog-A form for top-down modeling appears.

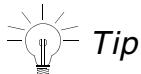


The check boxes that appear on the form depend on the cell type you select.

2. (Optional) Mark the *Dynamic Vdd sampling* check box to have the program sample Vdd continuously throughout the simulation.

If you do not mark this check box, the program samples Vdd once only, at the beginning of the simulation.

3. (Optional) Mark the *Add debug* check box to have the program add run-time debug information to the Verilog-A model (such as time points and parameter values).



As the simulator recognizes each input event, debug messages can provide you with information about why outputs changed and what the delays and slopes were.

4. (Optional) Mark the *Add comments* check box to have the program add comments describing the purpose of the Verilog-A code to the generated model.

5. (Optional) In the *Include file in header* field, do the following:

- a. Click ... to open the Choose File to Be Included in the Model Header form.
- b. Navigate to and select a file whose contents you want the program to include in the header of the generated Verilog-A behavioral model file.

The included file must have comment characters at the beginning of each line.

The program includes the contents of the specified file immediately following the first two lines.

6. (Optional) In the *Script output in header* field, do the following:
  - a. Click ... to open the Choose Script to Be Run with Output Included in the Model Header form.
  - b. Navigate to and select an executable script whose output you want the program to include in the generated Verilog-A behavioral model file.

The program includes the output immediately following the first two lines.
7. Click *OK*.

## Specifying Bottom-Up Modeling for AMS Models

To begin bottom-up modeling of AMS behavior models:

- Select the *Bottom-Up* tab to view the Bottom-Up page.

You can now do the following:

- [Specifying Bottom-Up Verilog-A\[MS\] Model Behaviors](#)
- [Specifying Bottom-Up VHDL-AMS Model Behaviors](#)
- [Specifying Bottom-Up Advanced Verilog-A Options](#)
- [Specifying Bottom-Up Verilog-D Model Options](#)

### Specifying Bottom-Up Verilog-A[MS] Model Behaviors

To specify model behaviors, do the following:

1. Select the *Verilog-A[MS]* tab on the Bottom-Up page.
2. In the *Behaviors* group box, select the behavior you want to model (such as timing or input capacitance).

The method you can use to model that behavior (such as the table method) and the parameters you can sweep appear in the next column.

For example, you might select *Timing model method* to specify the modeling method and parameters to sweep to model timing behavior.

3. In the method group box, do the following:
  - a. In the *Method* drop-down list, select a calibration method (such as *Table*).
  - b. Mark the check box for each parameter you want to sweep (such as *Temperature* or *Vdd voltage*) to model the behavior you selected.

**Note:** These parameters appear on the *Calibration* tab if you mark the *Calibrate* check box in the *Options* group box (see “[Specifying Bottom-Up AMS Model Options](#)”).

### Specifying Bottom-Up VHDL-AMS Model Behaviors

To specify model behaviors for VHDL-AMS, do the following:

1. Select the *AMS* tab on the Bottom-Up page.

2. In the *Behaviors* group box, select the behavior you want to model (such as timing or input capacitance).

The method you can use to model that behavior (such as the table method) and the parameters you can sweep appear in the next column.

For example, you might select *Timing model method* to specify the modeling method and parameters to sweep to model timing behavior.

3. In the method group box, do the following:

- a. In the *Method* drop-down list, select a calibration method (such as *Table*).
- b. Mark the check box for each parameter you want to sweep (such as *Temperature* or *Vdd voltage*) to model the behavior you selected.

**Note:** These parameters appear on the *Calibration* tab if you mark the *Calibrate* check box in the *Options* group box (see “[Specifying Bottom-Up AMS Model Options](#)”).

## Specifying Bottom-Up AMS Model Options

To specify AMS model options, do the following in the *Options* group box on the *Bottom-Up* page:

1. (Optional) Mark the *Calibrate* check box if you want the program to calibrate the model.

The *Calibration* tab becomes active. See [Calibrating Models](#) for more information.

2. (Optional) Mark the *Verify* check box if you want the program to verify the model.

3. (Optional) Mark check boxes for other options you want to enable.

For example, if you mark the *Minpulse* check box for the phase-frequency detector function, the program will characterize and model glitch generation on state changes.

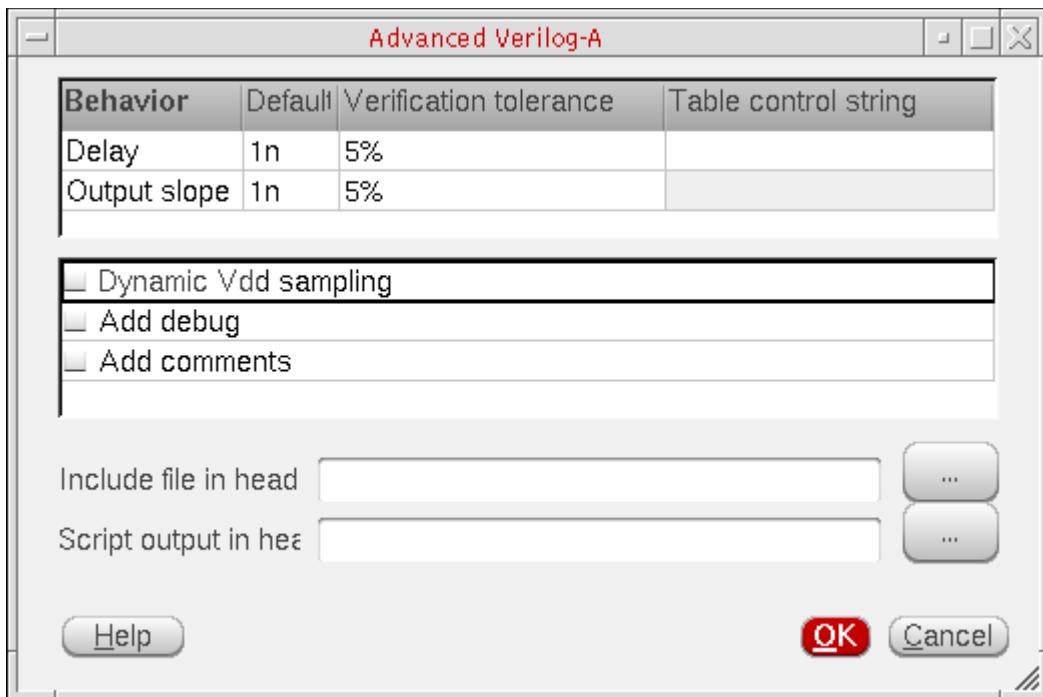
4. In the parameter table, click in the *Value* column to edit any option value you want to change.

## Specifying Bottom-Up Advanced Verilog-A Options

To specify advanced Verilog-A options for bottom-up modeling, do the following:

1. On the *Bottom-Up Verilog-A[MS] (AMS)* tab, click *Advanced*.

The Advanced Verilog-A form for bottom-up modeling appears.

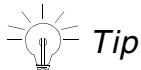


Default behaviors appear in the table at the top of the form.

The check boxes that appear on the form depend on the cell type you select.

2. (Optional) Click in the *Table control string* column and type a control string for a Spectre \$table\_model.

The control string specifies an interpolation method to use for delays and slopes (see "Interpolating with Table Models" in the *Cadence Verilog-A Language Reference*)



You can use error extrapolation (`E`) to make sure table index values are in the calibration range for a model. For example, a *Table control string* of `1E` specifies single-degree interpolation with error extrapolation at both ends of the table.

3. (Optional) Mark the *Dynamic Vdd sampling* check box to have the program sample Vdd continuously throughout the simulation.

If you do not mark this check box, the program samples Vdd once only, at the beginning of the simulation.

4. (Optional) Mark the *Add debug* check box to have the program add run-time debug information to the Verilog-A model (such as time points and parameter values).



*Tip*

As the simulator recognizes each input event, debug messages can provide you with information about why outputs changed and what the delays and slopes were.

5. (Optional) Mark the *Add comments* check box to have the program add comments describing the purpose of the Verilog-A code to the generated model.

6. (Optional) In the *Include file in header* field, do the following:

- a. Click ... to open the Choose File to Be Included in the Model Header form.
- b. Navigate to and select a file whose contents you want the program to include in the header of the generated Verilog-A behavioral model file.

The included file must have comment characters at the beginning of each line.

The program includes the contents of the specified file immediately following the first two lines.

7. (Optional) In the *Script output in header* field, do the following:

- a. Click ... to open the Choose Script to Be Run with Output Included in the Model Header form.
- b. Navigate to and select an executable script whose output you want the program to include in the generated Verilog-A behavioral model file.

The program includes the output immediately following the first two lines.

8. Click *OK*.

## Specifying Bottom-Up Modeling for Verilog-D

To begin bottom-up modeling of a Verilog-D behavioral model, do the following:

1. Select the *Verilog-D* tab on the *Bottom-Up* page
2. Select the *Create* check box.

**Note:** The *Create* check box is in the upper right corner of the tab.

The model parameters and options relevant for top-down design of the function you selected appear on the tab.

*Specify* appears in the *Method* drop-down list in the *Timing* group box as the timing model method. This method uses a Verilog *specify* block to model path delays for an accurate calibrated model.

3. (Optional) Click *Destination* to open the Model Destination form for specifying a library destination for the behavioral model the program generates.

See the following topics for more information:

- Specifying Bottom-Up Verilog-D Model Options on page 263
- Referencing User-Defined Primitives on page 264
- Adding Supply and Ground Sensitivity Attributes on page 264

## Specifying Bottom-Up Verilog-D Model Options

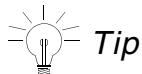
To specify Verilog-D model options, do the following in the *Options* group box on the *Bottom-Up Verilog-D* tab:

1. (Optional) Mark the *Calibrate* check box if you want the program to calibrate the model.  
The program uses single spot values to calibrate the Verilog-D model.
2. (Optional) Mark check boxes for options you want to enable.  
For example, mark the *Include timing constraints* check box to include timing constraints in the Verilog-D model.  
**Note:** This check box appears for sequential cells only (not combinational).
3. In the parameter table, click in the *Value* column to edit any option value you want to change such as *Default* (type a value for path delays).

## Referencing User-Defined Primitives

Sequential and some combinational cells (for example, MUX) reference a primitive file, `dcm/dcmName/dcmGen/cellName_prim.v`, in the project directory. When a design uses a calibrated Verilog-D model during Verilog or mixed-signal simulation (through a `config` to select the required views), you must make `cellName_prim.v` available to the simulator by setting the Verilog library search directory (`-y`) and library extension (`+libext`).

**Note:** The `cellName` comes from the design cell name on the *Design* tab.



### Tip

When you have several Verilog-D models, you should copy all of the files to a common directory and add a reference to that directory.

## Adding Supply and Ground Sensitivity Attributes

To add supply and ground sensitivity attributes, do the following:

- In the *Options* group box on the *Bottom-Up Verilog-D* tab, mark the *Add supply/ground sensitivity* check box.

The program adds the following attributes to all `input`, `output`, and `inout` pins in the generated Verilog-D model:

```
(* supplySensitivity = "supplyPin";
integer groundSensitivity = "groundPin"; *)
```

The program adds `supplySensitivity` and `groundSensitivity` attributes when you specify either of the following for low and high voltage data:

- Nothing (uses default values)

**Note:** Default values for `Vdd` and `Vss` pins are 3.0 V and 0 V, respectively.

- Referenced `Vdd` pin (high) or `Vss` pin (low)

When either of the following is specified, the application will issue a warning:

- A constant value
- An expression containing a supply voltage pin (for example, `Vdd-0.2` or `vpwr-0.4`)

If there are no supply pins (for example, because they are internal to the provided netlist/schematic design), the application does nothing and no attributes are added.

## Specifying Bottom-Up Modeling for Liberty

To begin bottom-up modeling of a Synopsys Liberty .lib model, do the following:

- On the *Liberty* tab of the *Bottom-Up* tab, mark the *Create* check box.

**Note:** The *Create* check box is in the upper right corner of the tab.

The model parameters and options relevant for top-down design of the function you selected appear on the tab.

See the following topics for more information:

- [Specifying Bottom-Up Liberty Model Behaviors on page 265](#)
- [Specifying Bottom-Up Liberty Model Options on page 265](#)
- [Generating a Liberty Library File from Individual Calibrated Cell Files on page 266](#)

### Specifying Bottom-Up Liberty Model Behaviors

To specify model behaviors, do the following:

1. In the *Behaviors* group box on the *Bottom-Up Liberty* tab, select the behavior you want to model (such as timing constraints).

The parameters you can sweep appear in the next column.

For example, you might select *Timing constraints model method* to specify the parameters to sweep to model timing constraints.

2. In the method group box (such as *Timing constraints*), mark the check box for each parameter you want to sweep (such as *Temperature* or *Vdd voltage*).

### Specifying Bottom-Up Liberty Model Options

To specify Liberty model options, do the following in the *Options* group box on the *Bottom-Up Liberty* tab:

1. (Optional) Mark check boxes for options you want to enable (such as *Acquire power* or *Use start/end conditions*).

For example, mark the *Use start/end conditions* check box to enable use of Liberty sdf\_cond\_start/when\_start for hold constraint conditions and sdf\_cond\_end/when\_end for setup constraint conditions.

**Note:** If you do not mark the *Use start/end conditions* check box, sdf\_cond/when

are used for setup and hold constraint conditions.

2. (Optional) Mark the *Calibrate* check box if you want the program to calibrate the model.

The *Calibration* tab becomes active. See “[Calibrating Models](#)” on page 269 for more information.

3. In the parameter table, click in the *Value* column to edit any option value you want to change.

For example, *Area*. *Area* is a cost number that the synthesis software uses when selecting cells. The *Area* number can be the laid-out cell, the number of transistors, or any other metric.

### **Generating a Liberty Library File from Individual Calibrated Cell Files**

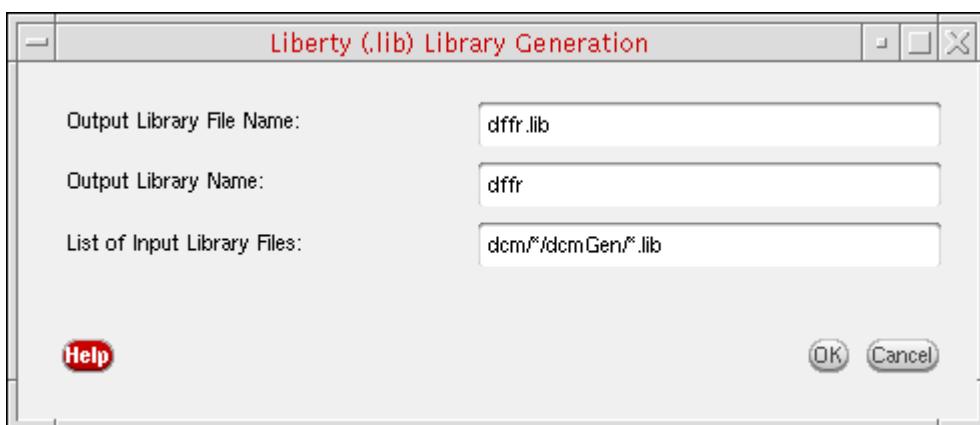
Once you have generated and calibrated several Liberty models, you can generate a Liberty library file from the set of calibrated cells.

**Note:** See also “[Generating Models and Running Tests](#)” on page 282 and “[Running Tests](#)” on page 283.

To generate a Liberty library file from several individual calibrated Liberty cell files, do the following:

1. Choose *Tools – Generate Liberty (.lib) library*.

The Liberty (.lib) Library Generation form appears.



2. In the *Output Library File Name* field, type a name for the Liberty library file.

The default is the DCM data file name prefix with a .lib extension. For example, if the DCM data file name is dff.dcm, the default output library file name is dff.lib.

- 3.** In the *Output Library Name* field, type a name for the Liberty library.

The default is the DCM data file name prefix. For example, if the DCM data file name is `dff.dcm`, the default output library name is `dff`.

**Note:** If you want to include library-specific information in the `library` section of the generated Liberty library file, you can create a file called `OutputLibraryName.inc` and put this file in the run directory. `OutputLibraryName` is the name you type in this field. For example, if you type `myCompiledLib` in the *Output Library Name* field, the program looks for a file called `myCompiledLib.inc`.

- 4.** In the *List of Input Library Files* field, type directory location and file name information for the calibrated Liberty cells.

**Note:** You can use asterisk (\*) as a wildcard.

For example:

`myDCM/myDesignCell/myLibertyFiles/*.lib`

The default is `dcm/*/dcmGen/*.lib`.

**Note:** If you have cell-specific information that you want to include in the `cell` sections of the generated Liberty library file, you can create files called `cellLibName.lib` and put them in the run directory. `cellLibName` is the prefix of each of the input library files. For each `cellLibName.lib` file the program finds, it looks for another file of the same prefix (`cellLibName`) with a `.inc` extension in the run directory. For example, if the program finds `nand2.lib` and `dff.lib`, it also looks for `nand2.inc` and `dff.inc`.

- 5.** Click *OK*.

The program processes the individual calibrated cell files (`.lib`) into a single library file, consolidating look-up tables and operating conditions.

## **Setting Up Model Calibration Requirements**

This section discusses how to set up model calibration requirements, and includes the following topics:

- [Calibrating Models](#) on page 269
- [Modifying Calibration Sweep Parameters](#) on page 271
- [Specifying Parameters](#) on page 278
- [Specifying Options](#) on page 281

## Calibrating Models

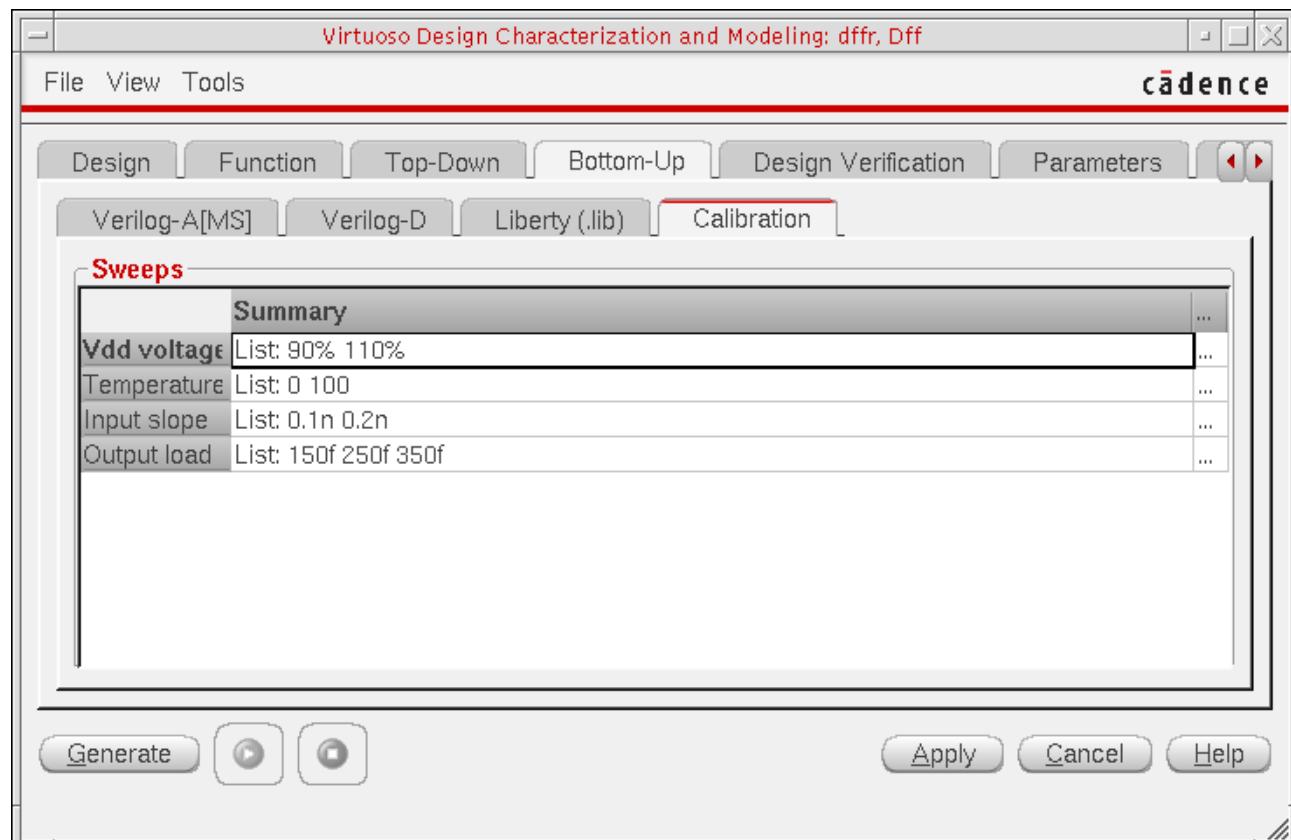


The function you select must support model calibration.

To calibrate a bottom-up model, do the following:

1. Specify bottom-up modeling for Verilog-A[MS], for Verilog-D, or for Liberty.
2. As part of the model specification, in the *Options* group box on the *Bottom-Up* tab for the model, mark the *Calibrate* check box  
(see “Specifying Bottom-Up AMS Model Options” on page 260,  
“Specifying Bottom-Up Verilog-D Model Options” on page 263, or  
“Specifying Bottom-Up Liberty Model Options” on page 265).
3. Select the *Calibration* tab.

The parameters you selected for calibration appear in the *Sweeps* group box on this tab.



4. (Optional) For each variable whose sweep parameters you want to change, click the ellipsis at the right end of the row.

The Sweep Setup form appears.

See “[Modifying Calibration Sweep Parameters](#)” on page 271 for more information.

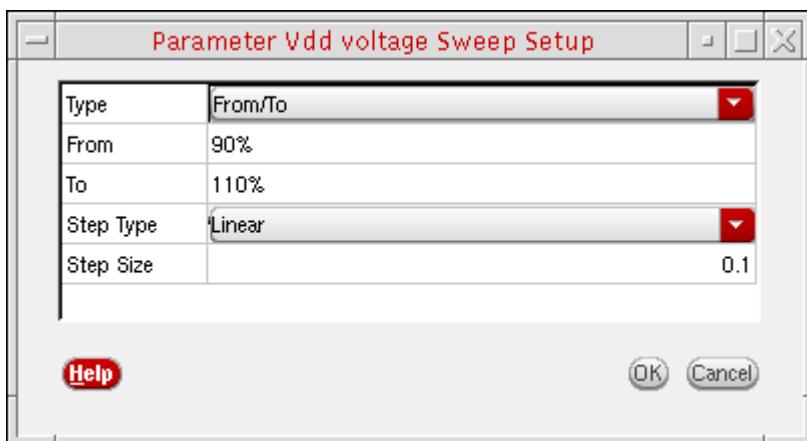
5. Click *Apply*.

## Modifying Calibration Sweep Parameters

You can specify the range and granularity of sweep variables that the program uses to calibrate bottom-up Verilog-A[MS] and Liberty models to original cell behavior. The sweep ranges define the operating range of the calibrated models. The program uses measured results from these sweeps to create tables to calibrate the models.

For each variable whose model calibration sweep parameters you want to change, do the following:

1. Click the ellipsis at the right end of the row.



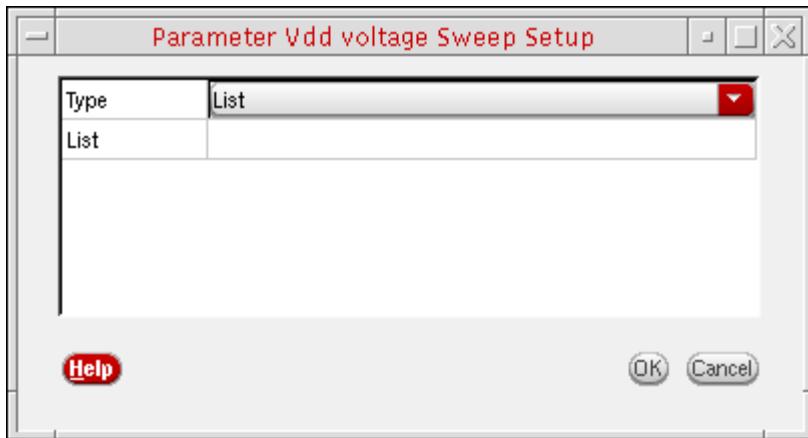
The Sweep Setup form appears.

2. In the *Type* drop-down list, select one of the following sweep types:

<u>List</u>	Specify a comma- or space-separated list of sweep values
<u>From/To</u>	Specify a sweep range and stepping information
<u>Center/Span</u>	Specify a center value and a span value for the sweep
<u>Center/Span%</u>	Specify a center value and a percentage span value

Relevant fields appear on the form.

## Specifying a List of Values



To specify a list of values for the model calibration sweep on the Sweep Setup form, do the following:

1. In the *List* field, type a list of values you want to include in your sweep.

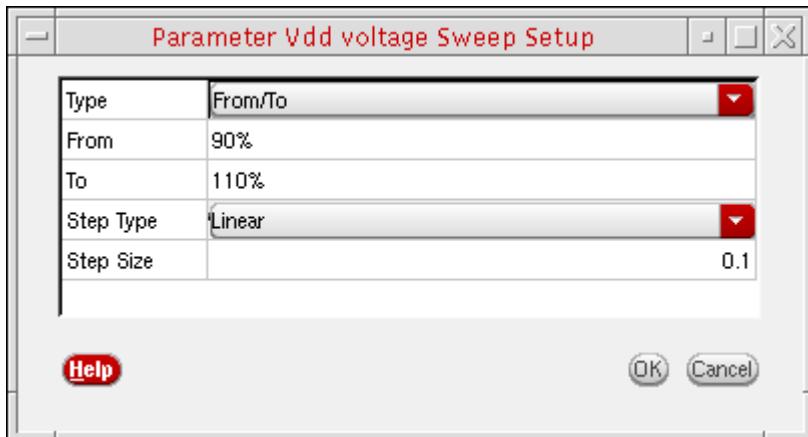
You can separate the values in the list with either a comma or a space.

2. Click *OK*.

Updated sweep summary information appears on the *Calibration* tab.

[Return](#)

## Specifying a Range of Values



To specify a range of values for the model calibration sweep on the Sweep Setup form, do the following:

1. In the *From* field, type the starting range value (or percentage for supply voltage).
2. In the *To* field, type the ending range value (or percentage for supply voltage).
3. In the *Step Type* drop-down list, select how you want the simulator to step from one value to the next:

<i>Linear</i>	Linear steps from the <i>From</i> value to the <i>To</i> value
<i>Decade</i>	Decade steps from the <i>From</i> value to the <i>To</i> value
<i>Octave</i>	Octave steps from the <i>From</i> value to the <i>To</i> value
<i>Logarithmic</i>	Logarithmic steps from the <i>From</i> value to the <i>To</i> value
<i>Times</i>	Steps taken according to a specified multiplier from the <i>From</i> value to the <i>To</i> value

4. Depending on the *Step Type* you selected, type the remaining value for your sweep specification in the field that appears:

<b>Step Type</b>	<b>Field that appears</b>	<b>Value</b>
<i>Linear</i>	<i>Step Size</i>	Size of steps to take when sweeping
<i>Decade</i>	<i>Steps/Decade</i>	How many steps to take per decade
<i>Octave</i>	<i>Steps/Octave</i>	How many steps to take per octave
<i>Logarithmic</i>	<i>Total Steps</i>	Total number of steps to take
<i>Times</i>	<i>Multiplier</i>	Multiplier for taking steps

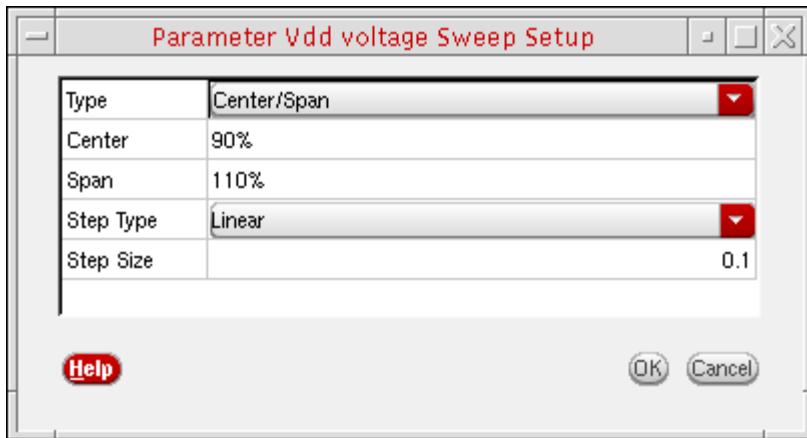
**Note:** Typing 10 for *Multiplier* is equivalent to selecting *Decade* and specifying 1 step per decade.

5. Click *OK*.

Updated sweep summary information appears on the *Calibration* tab.

[Return](#)

## Specifying a Center Value and Span Value



To specify a center value and a span value for the model calibration sweep on the Sweep Setup form, do the following:

1. In the *Center* field, type the center value (or percentage for supply voltage).
2. In the *Span* field, type a span value (or percentage for supply voltage).

The simulator can vary your parameter between *Center-Span* and *Center+Span* according to the *Step Type* you select (next).

3. In the *Step Type* drop-down list, select how you want the simulator to vary the parameter value:

<i>Linear</i>	Linear steps from <i>Center-Span</i> to <i>Center+Span</i>
<i>Decade</i>	Decade steps from <i>Center-Span</i> to <i>Center+Span</i>
<i>Octave</i>	Octave steps from <i>Center-Span</i> to <i>Center+Span</i>
<i>Logarithmic</i>	Logarithmic steps from <i>Center-Span</i> to <i>Center+Span</i>
<i>Times</i>	Steps taken according to a specified multiplier from <i>Center-Span</i> to <i>Center+Span</i>

4. Depending on the *Step Type* you selected, type the remaining value for your sweep specification in the field that appears:

Step Type	Field that appears	Value
<i>Linear</i>	<i>Step Size</i>	Size of steps to take when varying
<i>Decade</i>	<i>Steps/Decade</i>	How many steps to take per decade
<i>Octave</i>	<i>Steps/Octave</i>	How many steps to take per octave
<i>Logarithmic</i>	<i>Total Steps</i>	Total number of steps to take
<i>Times</i>	<i>Multiplier</i>	Multiplier for taking steps

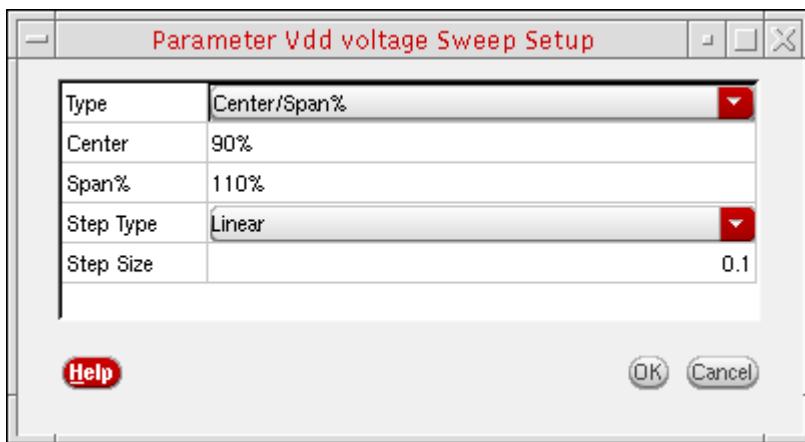
**Note:** Typing 10 for *Multiplier* is equivalent to selecting *Decade* and specifying 1 step per decade.

5. Click *OK*.

Updated sweep summary information appears on the *Calibration* tab.

[Return](#)

### Specifying a Center Value and Percentage Span Value



To specify a center value and a percentage span value for the model calibration sweep on the Sweep Setup form, do the following:

1. In the *Center* field, type the center value.

2. In the *Span* field, type a span value.

The simulator can vary your parameter between  $Center - \frac{(Span\% \times Center)}{100}$  and  $Center + \frac{(Span\% \times Center)}{100}$  according to the *Step Type* you select (next).

3. In the *Step Type* drop-down list, select how you want the simulator to vary the parameter value:

<i>Linear</i>	Linear steps from $Center - \frac{(Span\% \times Center)}{100}$ to $Center + \frac{(Span\% \times Center)}{100}$
<i>Decade</i>	Decade steps from $Center - \frac{(Span\% \times Center)}{100}$ to $Center + \frac{(Span\% \times Center)}{100}$
<i>Octave</i>	Octave steps from $Center - \frac{(Span\% \times Center)}{100}$ to $Center + \frac{(Span\% \times Center)}{100}$
<i>Logarithmic</i>	Logarithmic steps from $Center - \frac{(Span\% \times Center)}{100}$ to $Center + \frac{(Span\% \times Center)}{100}$
<i>Times</i>	Steps taken according to a specified multiplier from $Center - \frac{(Span\% \times Center)}{100}$ to $Center + \frac{(Span\% \times Center)}{100}$

4. Depending on the *Step Type* you selected, type the remaining value for your sweep specification in the field that appears:

<b>Step Type</b>	<b>Field that appears</b>	<b>Value</b>
<i>Linear</i>	<i>Step Size</i>	Size of steps to take when varying
<i>Decade</i>	<i>Steps/Decade</i>	How many steps to take per decade
<i>Octave</i>	<i>Steps/Octave</i>	How many steps to take per octave
<i>Logarithmic</i>	<i>Total Steps</i>	Total number of steps to take
<i>Times</i>	<i>Multiplier</i>	Multiplier for taking steps

**Note:** Typing 10 for *Multiplier* is equivalent to selecting *Decade* and specifying 1 step per decade.

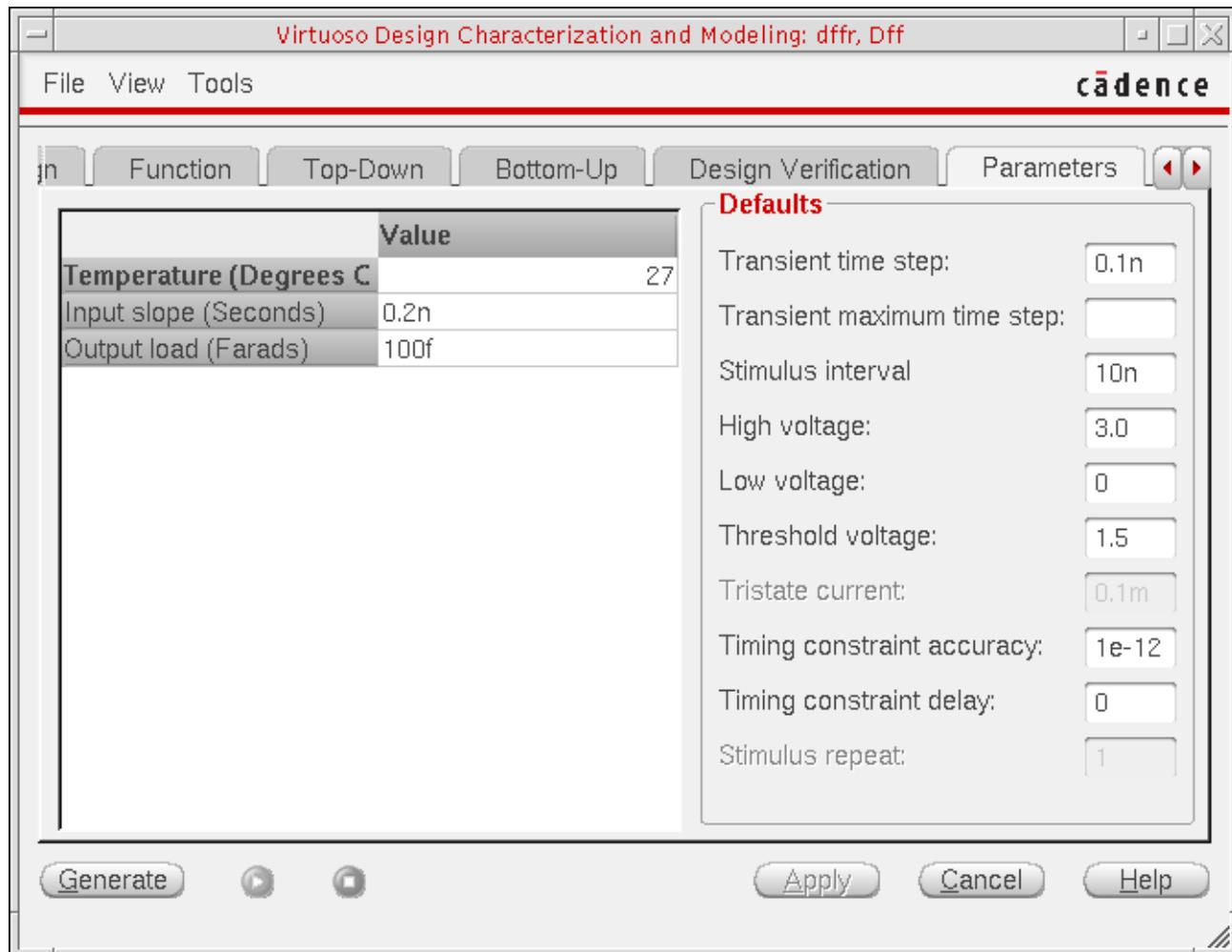
5. Click *OK*.

Updated sweep summary information appears on the *Calibration* tab.

[Return](#)

## Specifying Parameters

On the *Parameters* tab, you can specify parameter values and default values that apply to the behavior and simulation of the design function you selected. The parameters and default values that appear (on the left and right sides of the *Parameters* tab, respectively) depend on the cell type you selected.



Here are some parameters that might appear:

<i>Temperature</i>	Default simulation temperature (in degrees Centigrade)
<i>Input slope</i>	Slope or transition time for digital stimulus applied to input pins (in seconds)
<i>Output load</i>	Load capacitance for output pins (in Farads)

*Supply voltage*      Voltage connected to supply pins

The set of default values can include some or all of the following:

*Transient time step*      Default transient simulation time step (in seconds)  
*Transient maximum time step*      Maximum transient simulation time step (in seconds)  
*Stimulus interval*      Time between digital input stimulus edges (in seconds)  
*Low voltage*      Default low and high voltage values for digital input and output pins  
*High voltage*      For inputs, these values define the low and high driving voltages; for outputs, they determine the output threshold according to the following equation:

$$\frac{(\text{highVoltage} - \text{lowVoltage})}{2.0}$$

**Note:** The program uses these values if you do not specify low and high values for a design pin.

*Threshold voltage*      Default digital output voltage threshold  
**Note:** The program uses this value if you do not specify values for low and high voltages on a design pin.  
*Tristate current*      (Digital tristate outputs only) Current threshold below which an output is Z  
*Timing constraint accuracy*      Required accuracy for timing constraint measurements (+/- this value, in seconds)  
**Note:** The program uses this value to determine when to stop its bisection algorithm.

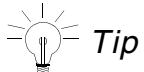
*Timing constraint delay*

Maximum delay difference allowed between the reference delay (measured at the maximum stimulus interval) and the delay measured at varying setup/hold/width times.

For example, a *Timing constraint delay* of 5 ns means that any delay value more than 5 ns different from the measured reference delay (measured at the maximum stimulus interval) violates this constraint. If the reference delay is 12 ns, a measured delay of 14 ns at 0.5 ns setup time does not violate this timing constraint, but a measured delay of 18 ns at 0.25 ns setup time does.

*Stimulus repeat*

Number of times to repeat a digital input stimulus



*Tip*  
You can increase this value if a cell has a problem initializing.

## Specifying Options

To specify options, do the following:

1. (Optional) In the *Generation* group box, mark the *Overwrite testbench schematic* check box to cause the program to overwrite the testbench schematic when you click Generate.

**Note:** You should not mark this check box if you have added additional components to the schematic that you do not want the program to overwrite.

2. (Optional) In the *Generation* group box, mark the *Overwrite ADEXL view* check box to cause the program to remove the old `adexl` view and create a new one when you click Generate.

**Note:** If you do not mark this check box, the program prompts you to confirm before overwriting the `adexl` view.

3. (Optional) In the *Simulation Options* group box, mark the *Plot waveforms* check box if you want the program to plot waveforms after simulating.

The program plots all the design pins and whatever you added to the test templates.

4. (Optional) In the *Job Control* group box, click *Setup* to open the Job Policy Setup form.

You can specify the settings you want the program to use for simulation jobs.

5. Click *Apply*.

## Generating Models and Running Tests

This section discusses generating models and tests and then running the tests, and includes the following topics:

- [Generating Tests and Models](#) on page 282
- [Running Tests](#) on page 283

### Generating Tests and Models

To generate tests and models (whatever you specified), do the following:

- Click *Generate*.

The program generates the items you specified (tests or models or both).

See also “[Viewing Generated Files](#)” on page 212.

What the program generates depends on the *Create* or *Calibrate* options you enable. See the following topics for more information:

- [Specifying Top-Down Modeling for AMS Models](#) on page 253
- [Specifying Bottom-Up Modeling for AMS Models](#) on page 259
- [Specifying Bottom-Up Modeling for Verilog-D](#) on page 263
- [Specifying Bottom-Up Modeling for Liberty](#) on page 265
- [Enabling Design Verification](#) on page 317
- [Calibrating Models](#) on page 269

See also “[Using the Tools Menu](#)” on page 210.

## Running Tests

To run tests, do the following:

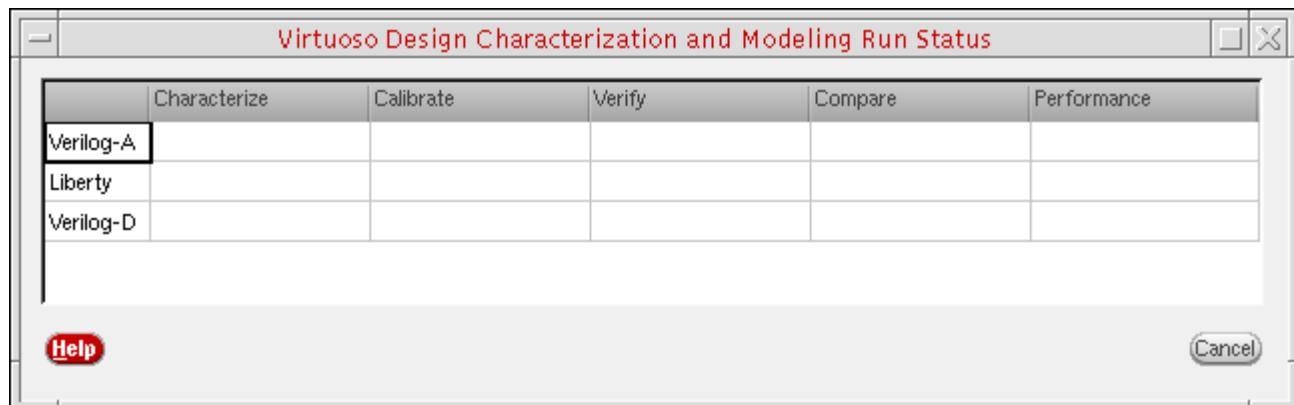
- Click the green Run button (or you can choose *Tools – Run All*).



The program performs the tasks you enabled for the design function you selected:

- ❑ For calibration, the program runs tests and sweeps to characterize, calibrate, verify, and compare the models to the design.
- ❑ For design verification, the program characterizes and compares the design to the design verification specifications.

The Run Status window appears.



Status keywords are

*Running* Task is running.

*Cancelled* You cancelled the task.

*Done* Task is finished.

*Failed* Task failed. You can double-click to see more information.

See also

- [Viewing Run Status on page 284](#)
- [Viewing Run Status Using the View Menu on page 286](#)

- [Stopping the Run](#) on page 286

See also “[Viewing Generated Files](#)” on page 212.

### Viewing Run Status

The Run Status window appears when you click the green Run button in the Virtuoso Design Characterization and Modeling window (or when you choose [View – Run Status](#)).



The program reports status for any task you enabled that requires simulation. (The program does not report run status for top-down modeling or bottom-up models for which you have not enabled calibration.)

When the program is performing a task, *Running* appears in the column.

	Characterize	Calibrate	Verify	Compare	Performance
Design Verification	Running				
Verilog-A	Running				
Liberty	Running				
Verilog-D	Running				

When the program is done performing a task, *Done* appears in the column.

	Characterize	Calibrate	Verify	Compare	Performance
Design Verification	Running				
Verilog-A	Done (725s)	Running			
Liberty	Running				
Verilog-D	Done (659s)	Done			

## Virtuoso Analog Design Environment GXL User Guide

### Model Generation

For characterization and verification of the model, the program reports the duration of the task, in clock seconds (not pure simulation time).

	Characterize	Calibrate	Verify	Compare	Performance
Design Verification	Running				
Verilog-A	Done (725s)	Done	Running		
Liberty	Running				
Verilog-D	Done (659s)	Done			

[Help](#)

[Cancel](#)

**Note:** In addition to the simulation time, task-duration time includes setup (such as netlisting) and post-processing time. The accuracy of this number can be affected by jobs that run on different machines.

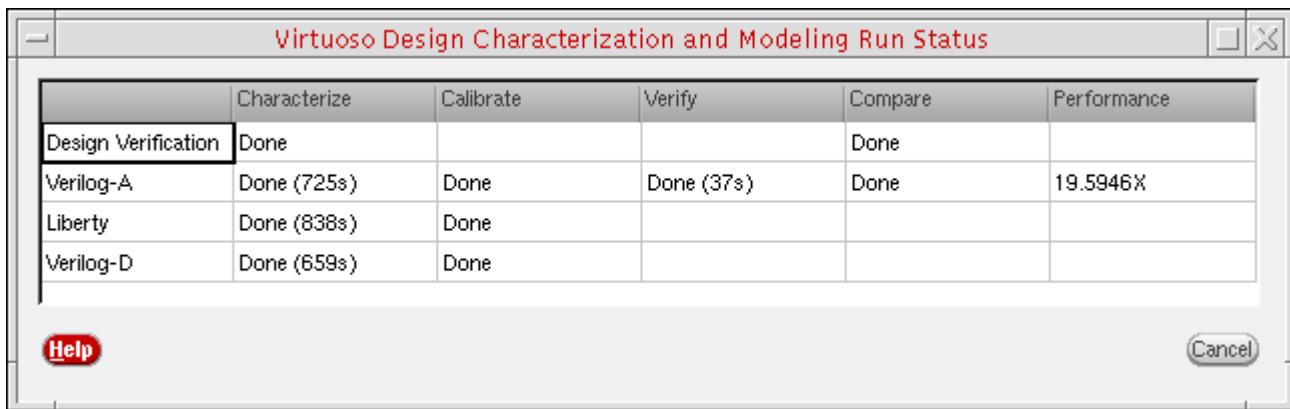
	Characterize	Calibrate	Verify	Compare	Performance
Design Verification	Done			Done	
Verilog-A	Done (725s)	Done	Done (37s)	Done	19.5946X
Liberty	Done (838s)	Running			
Verilog-D	Done (659s)	Done			

[Help](#)

[Cancel](#)

For a model that you both characterize and verify, the program reports the relative performance of these tasks in the *Performance* column:

$$\frac{\text{CharacterizationTime}}{\text{VerificationTime}} = \text{Performance}$$



### **Viewing Run Status Using the View Menu**

To view run status using the *View* menu, do the following:

- Choose *View – Run Status*.  
The Run Status window appears

### **Stopping the Run**

To stop running tests, do the following:

- Click the Stop button (which is only active while simulations are running).



The program stops running the tests. Any currently running jobs appear as *Cancelled*.

## Generating Liberty MS Models

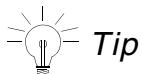
You can use DCM to generate Liberty (.lib) mixed-signal models for both digital flows, which require .lib models for all blocks, and for mixed-signal blocks, which may have behaviors that can be represented using the Liberty (.lib) model format.

Liberty models are constructed from individual configured and calibrated items, such as delay paths, pin capacitances, timing constraints, etc. This methodology does not require that you have created an existing cell type for a specific design function. The entire contents of the model are created by adding and configuring the individual components, thus allowing models of arbitrary complexity to be supported. These models can be calibrated using specified values or measured results from an ADE XL result database.

In DCM, you first either input an existing design and cell type or create a new design. You then add and set up individual model components. The calibration values for these model components can be directly entered, or they can use measured results, either single values or swept values for creating tables and K factors. The model is then generated and calibrated from the model components, with no design-specific IP development required.

To generate Liberty (.lib) mixed-signal models:

1. Launch DCM in Black Box Liberty .lib MS mode.  
For more information, see [Launching DCM](#).
2. Select the *Design* tab and specify design information.  
For more information, see [Specifying a Design](#).
3. Select the *Function* tab and set up function and pin types.  
For more information, see [Specifying a Function](#).

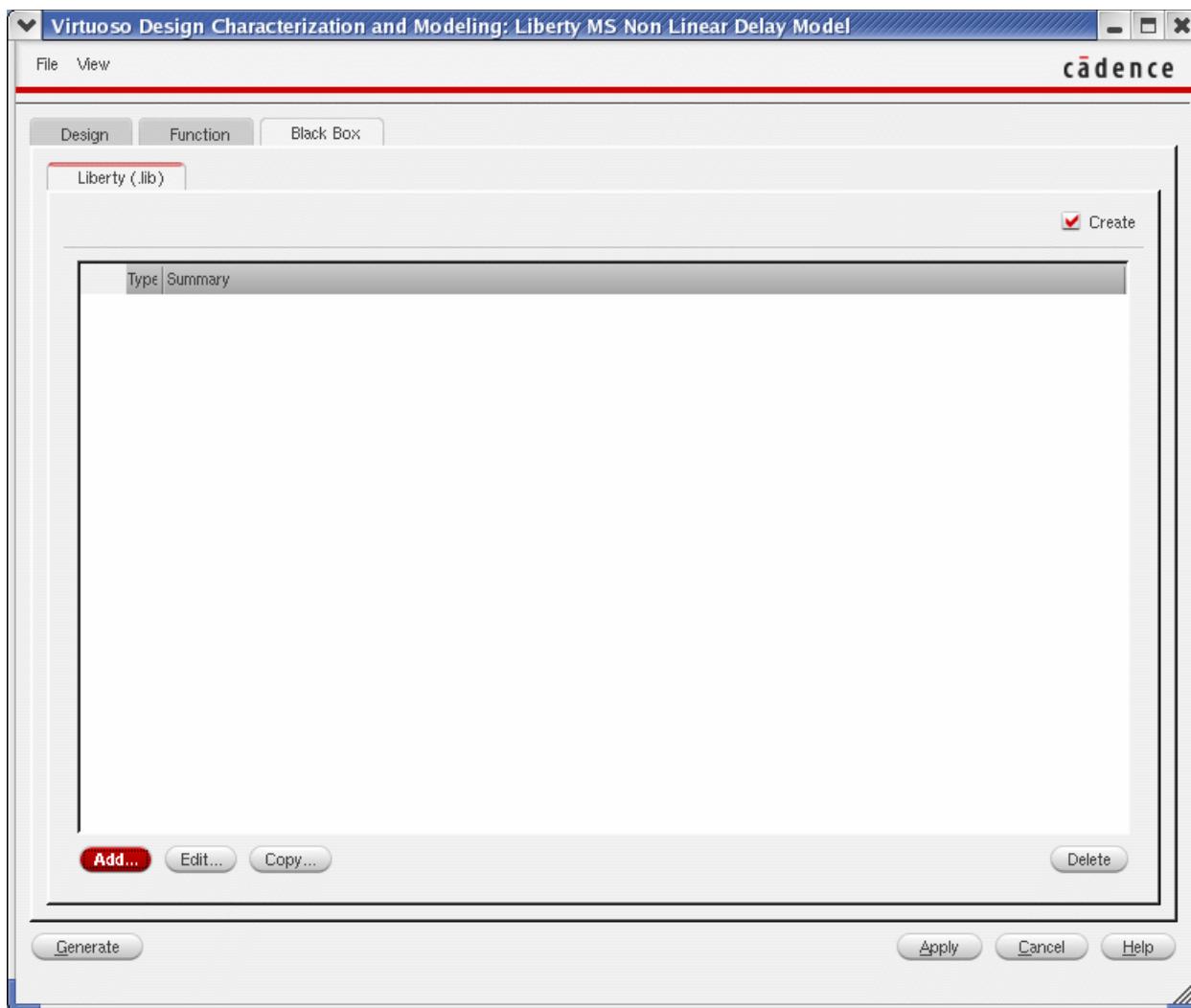


Default pin types are set based on the specified design.

## Virtuoso Analog Design Environment GXL User Guide

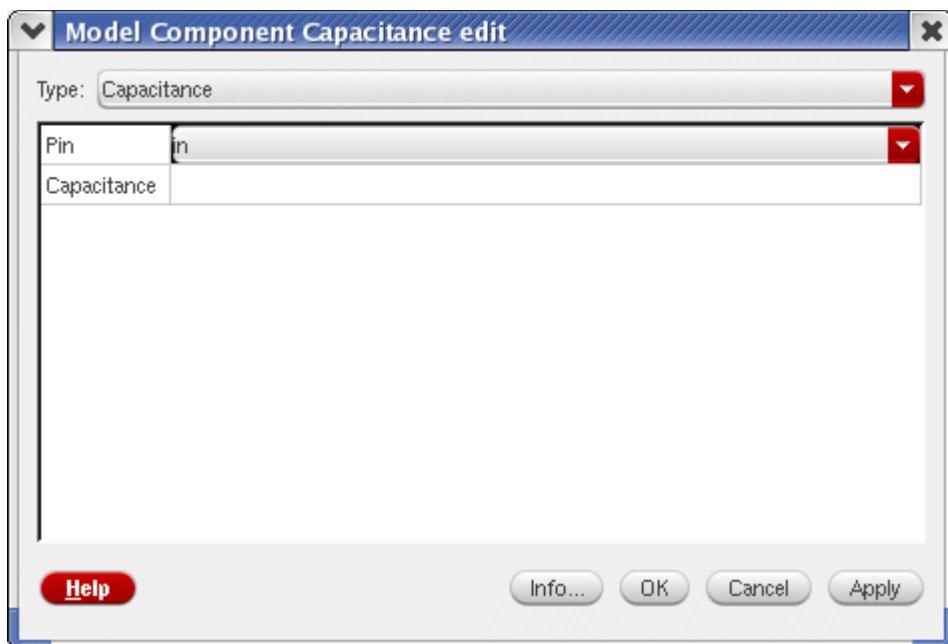
### Model Generation

4. Select the *Black Box* tab.



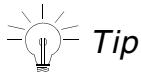
5. Click *Add* to add a new model component.

The Model Component Capacitance form appears.



6. Choose a type of model from the *Type* cyclic field.

The display of the list box changes based on the type selected.



You can always click *Info* for more information.

7. Specify values for the options for this model component.

For more information, see [Model Component Parameters](#), below.

8. Click *Apply* to check and save your data.

The new model component is listed in the Black Box page in the main DCM form.

9. Repeat [step 6](#) through [step 8](#) for all models.

10. Click *OK* to dismiss the Model Component Capacitance form.

All model components are listed in the Black Box page.

11. Click *Apply* to save your data.

12. In the main DCM form, click *Generate* to create and calibrate the .lib model.

## Model Component Parameters

The following tables list the parameters for the current set of model components for Liberty .lib MS models.

**Table 7-1 Capacitance**

Parameter	Description
Pin	Choose a pin name from the drop-down list.
Capacitance	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values for the capacitance.

**Table 7-2 Conditional Pulse Width**

Parameter	Description
Pin	Choose a pin name from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.
SDF condition	Enter a condition to be set in the SDF files.
Low Pulse Width	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values for the low pulse width.
High Pulse Width	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values for the high pulse width.

**Table 7-3 Non Linear Delay**

Parameter	Description
From Pin	Choose a pin name from the drop-down list.
From Edge	Choose an edge name from the drop-down list.
To Pin	Choose a pin name from the drop-down list.
Timing Sense	Choose a timing sense from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.
SDF condition	Enter a condition to be set in the SDF files.

**Table 7-3 Non Linear Delay**

<b>Parameter</b>	<b>Description</b>
Cell Rise	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the cell rise delay.  The cell rise delay is the time elapsed from the point at which the “from” pin goes through its threshold until the point at which “to” pin goes through its threshold.
Rise Transition	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the output slope of the rise transition.  The output slope of the rise transition is based on the time taken for the “To” pin to go from low to high.
Cell Fall	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for cell fall.
Fall Transition	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the output slope of the fall transition.  The output slope of the fall transition is based on the time taken for the “To” pin to go from high to low.

**Table 7-4 Non Linear Disable (?Z) Delay**

<b>Parameter</b>	<b>Description</b>
From Pin	Choose a pin name from the drop-down list.
To Pin	Choose a pin name from the drop-down list.
Timing Sense	Choose a timing sense from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.
SDF condition	Enter a condition to be set in the SDF files.

**Table 7-4 Non Linear Disable (?Z) Delay**

Parameter	Description
Cell Rise (0Z)	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the cell rise delay.  The cell rise delay is the time elapsed from the point at which the “from” pin goes through its threshold until the point at which “to” pin goes through its threshold.
Cell Fall	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for cell fall.

**Table 7-5 Non Linear Enable (Z?) Delay**

Parameter	Description
From Pin	Choose a pin name from the drop-down list.
To Pin	Choose a pin name from the drop-down list.
Timing Sense	Choose a timing sense from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.
SDF condition	Enter a condition to be set in the SDF files.
Cell Rise (Z1)	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the cell rise delay.  The cell rise delay is the time elapsed from the point at which the “from” pin goes through its threshold until the point at which “to” pin goes through its threshold.
Rise (Z1) Transition	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the output slope of the rise transition.  The output slope of the rise transition is based on the time taken for the “To” pin to go from low to high.

**Table 7-5 Non Linear Enable (Z?) Delay**

Parameter	Description
Cell Fall (Z0)	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for cell fall.
Fall (Z0) Transition	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the output slope of the fall transition.  The output slope of the fall transition is based on the time taken for the “To” pin to go from high to low.

**Table 7-6 Non Linear Hold**

Parameter	Description
Data Pin	Choose a pin name from the drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.
SDF condition	Enter a condition to be set in the SDF files.
Rise Constraint	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the rise constraint.

**Table 7-7 Non Linear Recovery**

Parameter	Description
Level Pin	Choose a pin name from the drop-down list.
Level Active	Choose an activity level from the Level Active drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
Clock Edge	Choose a clock edge from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.

**Table 7-7 Non Linear Recovery**

Parameter	Description
SDF condition	Enter a condition to be set in the SDF files.
Recover	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for recovery.

**Table 7-8 Non Linear Setup**

Parameter	Description
Data Pin	Choose a pin name from the drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
Clock Edge	Choose a clock edge from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.
SDF condition	Enter a condition to be set in the SDF files.
Rise Constraint	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the rise constraint.
Fall Constraint	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the fall constraint.

**Table 7-9 Pulse Width**

Parameter	Description
Pin	Choose a pin name from the drop-down list.
State	Choose a state from the drop-down list.
Pulse Width	Select to <u>choose the results from an ADE XL run</u> , then specify the test and result values and variable types and values for the pulse width.

**Table 7-10 Area**

<b>Parameter</b>	<b>Description</b>
Area	Enter a value for the area.

**Table 7-11 Design Rule Capacitance**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Min. Capacitance	Enter a minimum capacitance value in pF.
Max. Capacitance	Enter a maximum capacitance value in pF.

**Table 7-12 Design Rule Fanout**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Min. Fanout	Enter a minimum fanout value.
Max. Fanout	Enter a maximum fanout value.

**Table 7-13 Design Rule Fanout Load**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Fanout Load	Enter a fanout load value.

**Table 7-14 Design Rule Max Transition for Input Pin**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Max Transition	Enter a maximum transition value in ns.

**Table 7-15 Design Rule Transition**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Min. Transition	Enter a minimum transition value in ns.
Max. Transition	Enter a maximum transition value in ns.

**Table 7-16 Flip Flop Function**

<b>Parameter</b>	<b>Description</b>
Clock Boolean Expression	Enter a value for the clock boolean expression.
Slave Clock Boolean Expression	Enter a value for the slave clock boolean expression.
Data Boolean Expression	Enter a value for the data boolean expression.
Clear Boolean Expression	Enter a value for the clear boolean expression.
Preset Boolean Expression	Enter a value for the preset boolean expression.
Q state when Preset and Clear active	Choose a value from the drop-down list for the Q state when Preset and Clear are active.
QB state when Preset and Clear active	Choose a value from the drop-down list for the QB state when Preset and Clear are active.
Data Boolean Expression type	Choose a value from the drop-down list for the data boolean expression type.

**Table 7-17 Pin Function**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Boolean Expression	Enter a value for the boolean expression.

**Table 7-18 Latch Function**

<b>Parameter</b>	<b>Description</b>
Enable Boolean Expression	Enter a value for the enable boolean expression.
Data Boolean Expression	Enter a value for the data boolean expression.
Clear Boolean Expression	Enter a value for the clear boolean expression.
Preset Boolean Expression	Enter a value for the preset boolean expression.
Q state when Preset and Clear active	Choose a value from the drop-down list for the Q state when Preset and Clear are active.
QB state when Preset and Clear active	Choose a value from the drop-down list for the QB state when Preset and Clear are active.

**Table 7-19 Three State (Z) Disable Function**

<b>Parameter</b>	<b>Description</b>
Pin	Choose a pin name from the drop-down list.
Disable Boolean Expression	Enter a value for the disable boolean expression.

**Table 7-20 Generic Delay**

<b>Parameter</b>	<b>Description</b>
From Pin	Choose a pin name from the drop-down list.
From Edge	Choose an edge name from the drop-down list.
To Pin	Choose a pin name from the drop-down list.
Timing Sense	Choose a timing sense from the drop-down list.
When condition	Enter a condition for when you want this model should be used. Define this condition in Liberty (.lib) syntax.

**Table 7-20 Generic Delay**

Parameter	Description
SDF condition	Enter a condition to be set in the SDF files.
Intrinsic Rise	Enter a value in ns for the intrinsic rise.
Rise Resistance	Enter a value in ns/pF for the rise resistance.
Intrinsic Fall	Enter a value in ns for the intrinsic fall.
Fall Resistance	Enter a value in ns/pF for the fall resistance.

**Table 7-21 Generic Disable (?Z) Delay**

Parameter	Description
From Pin	Choose a pin name from the drop-down list.
To Pin	Choose a pin name from the drop-down list.
0 to Z disable	Enter a value in ns for the 0 to Z disable value.
1 to Z disable	Enter a value in ns for the 1 to Z disable value.

**Table 7-22 Generic Enable (Z?) Delay**

Parameter	Description
From Pin	Choose a pin name from the drop-down list.
To Pin	Choose a pin name from the drop-down list.
0 to Z disable	Enter a value in ns for the 0 to Z enable value.
1 to Z disable	Enter a value in ns for the 1 to Z enable value.

**Table 7-23 Generic Hold**

Parameter	Description
Data Pin	Choose a pin name from the drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
Clock Edge	Choose a clock edge from the drop-down list.

**Table 7-23 Generic Hold**

<b>Parameter</b>	<b>Description</b>
Intrinsic Rise	Enter a value in ns for the intrinsic rise.
Intrinsic Fall	Enter a value in ns for the intrinsic fall.

**Table 7-24 Generic Recovery**

<b>Parameter</b>	<b>Description</b>
Level Pin	Choose a pin name from the drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
Clock Edge	Choose a clock edge from the drop-down list.
Intrinsic Rise	Enter a value in ns for the intrinsic rise.
Intrinsic Fall	Enter a value in ns for the intrinsic fall.

**Table 7-25 Generic Removal**

<b>Parameter</b>	<b>Description</b>
Level Pin	Choose a pin name from the drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
Clock Edge	Choose a clock edge from the drop-down list.
Intrinsic Rise	Enter a value in ns for the intrinsic rise.
Intrinsic Fall	Enter a value in ns for the intrinsic fall.

**Table 7-26 Generic Setup**

<b>Parameter</b>	<b>Description</b>
Data Pin	Choose a pin name from the drop-down list.
Clock Pin	Choose a pin name from the drop-down list.
Clock Edge	Choose a clock edge from the drop-down list.
Intrinsic Rise	Enter a value in ns for the intrinsic rise.

## Virtuoso Analog Design Environment GXL User Guide

### Model Generation

---

**Table 7-26 Generic Setup**

Parameter	Description
Intrinsic Fall	Enter a value in ns for the intrinsic fall.

---

## Calibrating Verilog-A[MS] Models

---

You can use the Model Calibration feature of Virtuoso® Design Characterization and Modeling (DCM) to calibrate Verilog-A[MS] models. The Model Calibration tool is used to generate a silicon-calibrated behavioral model for a transistor-level design. A behavioral model is a high-level description that models the behavior of a mixed-signal function or design, while a calibrated model is a behavioral model that uses characterized results to define parameter values. Characterized results are provided in one of the following ways:

- Run simulations and sweep parameters of the transistor-level circuit to generate a look-up table of calibrated values.
- Substitute a constant, calibrated value for a parameter, obtained as a result of simulation or by some other means

There are four steps to calibrating your model. First, you specify the source for the model, either a lib/cell/view or a file name. Next, you associate parameters and table calls with calibration requirements and values/results. Then you specify a lib/cell/view or file name for the location of the resulting model. Finally, after the tool reads the source model, you calibrate to set up the model and save it to the specified destination lib/cell/view or file name.

You can specify a lib/cell/view for your source model and a file for the resulting model, which is useful for calibrating bmslib, etc. You can also specify a file name for your source model and a lib/cell/view for the resulting model, which is helpful for adding new models to a library.

For more information, see:

- [Launching Model Calibration](#)
- [Specifying the Source Model](#)
- [Setting Up the Model](#)
- [Specifying the Destination Model](#)
- [Saving the Model Calibration Setup](#)
- [Calibrating the Model](#)

## Launching Model Calibration

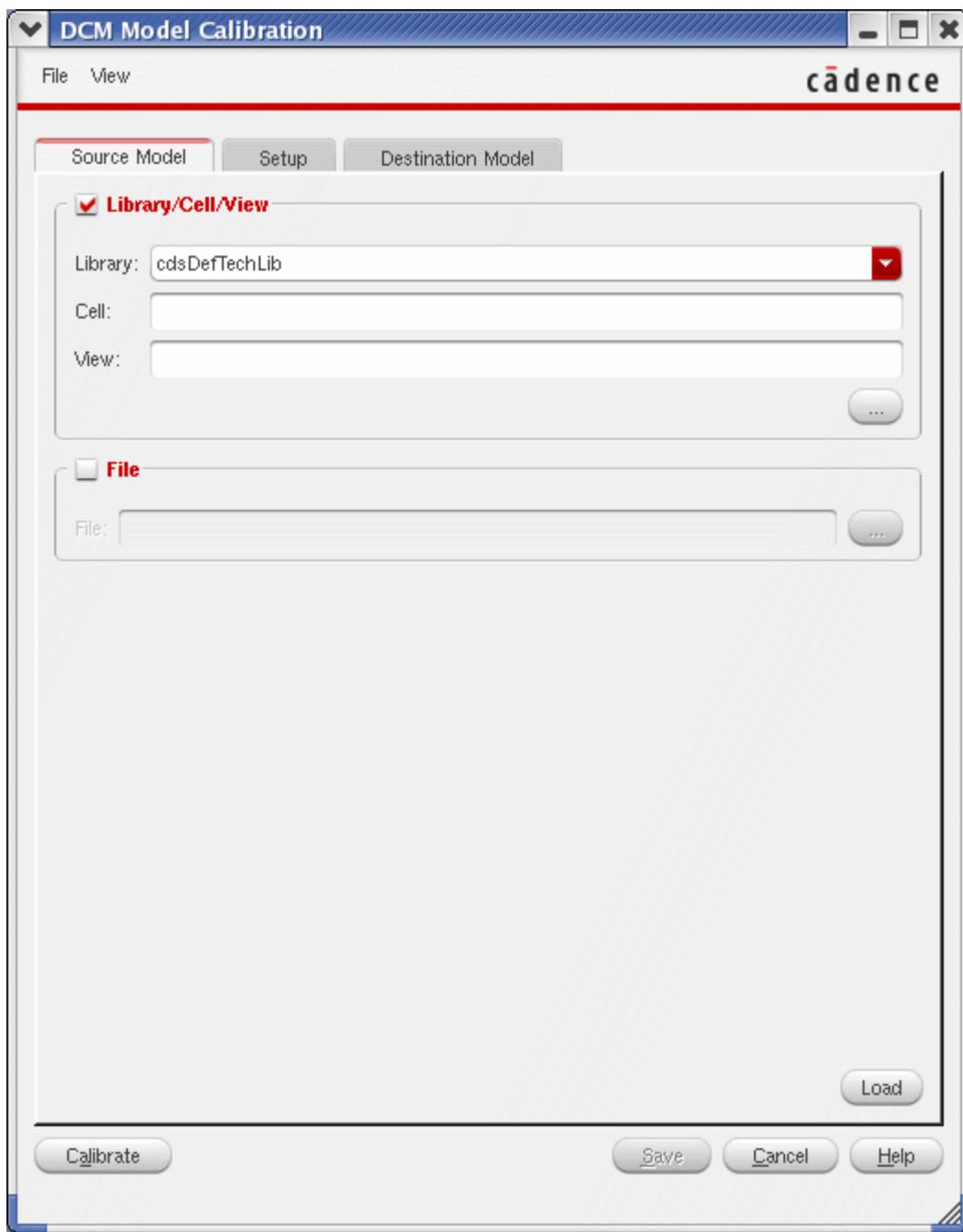
To begin calibrating your model:

- Choose *Tools—Characterization and Modeling—Verilog-A[MS] Model Calibration* from the CIW.

## Virtuoso Analog Design Environment GXL User Guide

### Calibrating Verilog-A[MS] Models

The DCM Model Calibration form appears.



## Specifying the Source Model

You begin model calibration by specifying the source model.

1. If necessary, select the Source Model tab.
2. Select one of the following check boxes that defines the location of the source model:
  - Library/Cell/View
  - File
3. Specify the source model location:

For a Library/Cell/View, do one of the following:

- Choose the library from the pulldown menu, then enter the cell and view names in the Cell and View field.
- Click ... to browse for the lib/cell/view.

For a File, do one of the following:

- Enter the path and file name in the File field.
- Click ... to browse for the file location.

The Model Calibration form automatically displays the Setup page.

## **Setting Up the Model**

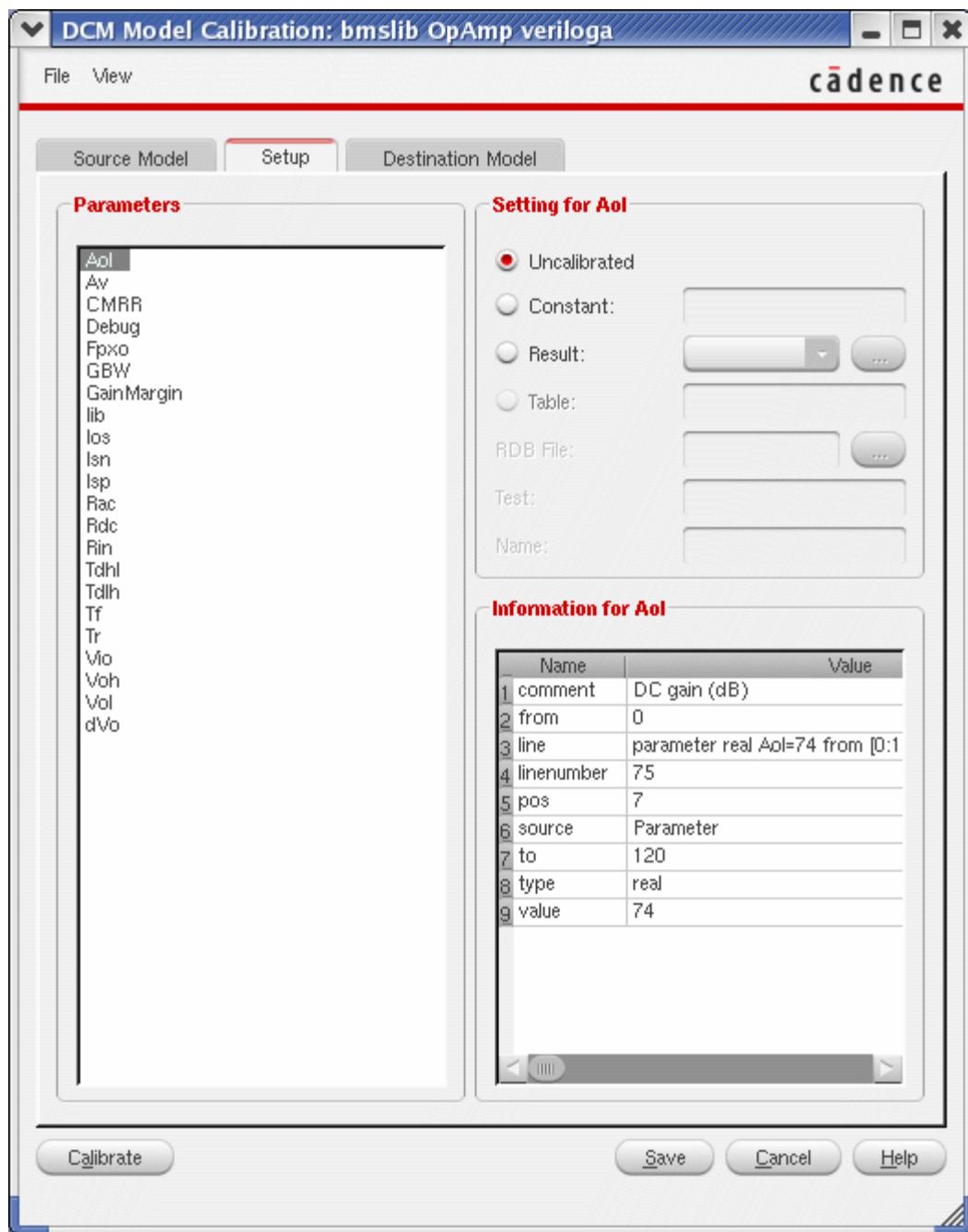
In the Setup tab, you associate parameters and table calls with calibration requirements and values/results.

1. If necessary, select the Setup tab.

# Virtuoso Analog Design Environment GXL User Guide

## Calibrating Verilog-A[MS] Models

The Setup page appears.



2. In the Parameters list box, select a parameter you want to associate with requirements and results.
3. In the Setting for <parameter> pane, select a radio button that describes the setting for this parameter, then specify the required information.

- Uncalibrated
- Single Value Constant
- Result
- Table

**4. In the Information for <parameter> table, modify the information as desired.**

This table displays all information about the selected parameter or table call that was extracted from the model.

### **Single Value Constant**

If the parameter should be passed to the model as a single value:

1. Select the *Constant* radio button.
2. Specify a value in the field.

### **Single Value from a Measured Result**

If the parameter should be passed to the model as a measured value from an ADE XL results database:

1. Select the *Result* radio button.
2. Click ... to browse for an ADE XL results database.



3. Browse for the results using one of the following methods:

**To use the last selected ADE XL cell view:**

- a. Select the *Current ADE XL Lib/Cell/View* radio button.

The dropdown list displays all previously loaded ADE XL cell views and run numbers.

- b. Choose an ADE XL lib/cell/view from the dropdown list.
- c. To utilize a run from the ADE XL cell view other than the one listed, click ....

The History Selection form appears.



- d. From the dropdown list, choose the run from which you want to use the results.
- e. Click *OK*.

**To select the results from a run in an ADE XL view:**

- a. Select the *ADE XL Lib/Cell/View* radio button.

- b. Click ... to browse for the ADE XL cell view.

You can also enter the lib/cell/view then click *Load*.

- c. Browse to the ADE XL cell view you want to use, then click *OK*.

The History Selection form appears.



- d. From the dropdown list, choose the run from which you want to use the results.
- e. Click *OK*.

**To select a results database file:**

- a. Select the *File* radio button.
  - b. Click ... to browse for the results database.
  - c. Browse to the results database file you want to use, then click *Open*.
4. To select the test to be used to obtain the measured result for model calibration, choose a test name from the *Test* dropdown list. To utilize the default listed Test, click *Load*.
  5. To select a measured result to use for model calibration, choose a result from the *Result* dropdown list.
  6. Click *OK*.

The Results database is listed in the Setting for <parameter> pane.

## Look-Up Table

If this parameter is used to create a .vat table file whose name is used in a \$table\_model function call in the behavioral model:

1. Select the *Table* radio button.
2. Click ... to browse for an ADE XL results database.



3. Browse for the results using one of the following methods:

### To use the last selected ADE XL cell view:

- a. Select the *Current ADE XL Lib/Cell/View* radio button.
- b. To utilize a run from the ADE XL cell view other than the currently-listed one, click ....

The History Selection form appears.



- c. From the dropdown list, choose the run from which you want to use the results.
- d. Click *OK*.

**To select the results from a run in an ADE XL view:**

- a. Select the *ADE XL Lib/Cell/View* radio button.
- b. Click ... to browse for the ADE XL cell view.
- c. Browse to the ADE XL cell view you want to use, then click *OK*.

The History Selection form appears.



- d. From the dropdown list, choose the run from which you want to use the results.
- e. Click *OK*.

**To select a results database file:**

- a. Select the *File* radio button.
  - b. Click ... to browse for the results database.
  - c. Browse to the results database file you want to use, then click *Open*.
4. To select the test to be used to obtain the measured result for model calibration, choose a test name from the *Test* dropdown list.
  5. To select a measured result to use for model calibration, choose a result from the *Result* dropdown list.

**6. Click *OK*.**

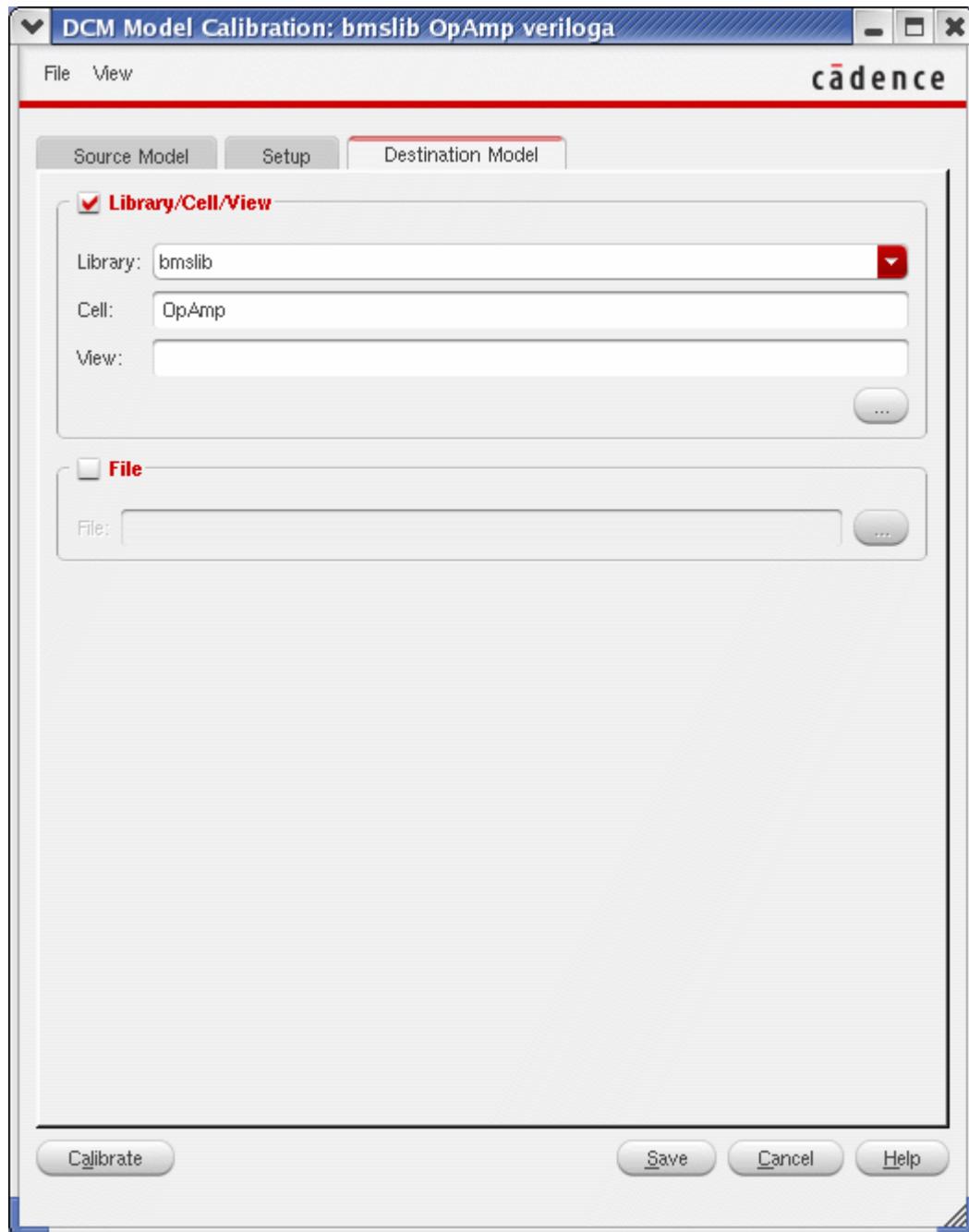
The Results database is listed in the Setting for <parameter> pane.

## Specifying the Destination Model

On the Destination tab, you can specify the location of the new model.

1. Select the Destination Model tab.

The Destination Model page appears.



**2.** Select one of the following check boxes that defines the new location of the model:

- Library/Cell/View
- File

**3.** Specify the destination model location:

For a Library/Cell/View, do one of the following:

- Choose the library, cell, and view from the pulldown menus.
- Click ... to browse for the lib/cell/view.

For a File, do one of the following:

- Enter the path and file name in the File field.
- Click ... to browse for the file location.

## Saving the Model Calibration Setup

When you have finished setting up your model calibration requirements, you can save your model calibration setup data to a .dmc (DCM Model Calibration) file. You can reload this file to recalibrate this model in the future.

To save your model calibration setup data:

1. Click *Save*.

The Save model calibration form appears.

2. Browse to the desired location and specify a file name in the File name field.
3. Click *Save*.

## Calibrating the Model

Once you have set up your model and saved the data, you can calibrate the model.



***If you have not yet saved your model calibration setup information, you will be prompted to save it prior to calibration.***

To calibrate the model:

- Click *Calibrate*.

The model is calibrated and saved to the specified Destination location.

**Virtuoso Analog Design Environment GXL User Guide**  
Calibrating Verilog-A[MS] Models

---

---

# Design Verification

---

You can verify a design function against a set of specifications using the Design Characterization and Modeling (DCM) feature of the Virtuoso® Analog Design Environment GXL (ADE GXL).

See the following topics for more information:

- [Specifying a Design](#) on page 234
- [Specifying a Function](#) on page 241
- [Enabling Design Verification](#) on page 317
- [Defining Specifications for Design Verification](#) on page 318
- [Specifying Conditions for Design Verification](#) on page 320
- [Specifying Parameters](#) on page 278
- [Specifying Options](#) on page 281

**Note:** See [Chapter 7, “Model Generation”](#) for information about DCM’s model generation capabilities.

## Enabling Design Verification

To enable design verification, do the following:

- On the *Design Verification* tab, mark the *Create* check box.

Tests and measurements for design verification appear on the *Specifications* tab. The program derives this information from the design, function and pin data you provide on the *Design* and *Function* tabs. You can specify conditions for design verification on the *Conditions* tab.

## Defining Specifications for Design Verification

You can define minimum or maximum (or both) test measurement values on the *Specifications* tab. The *Specifications* tab is a subtab of the [Design Verification](#) tab.

The screenshot shows a software interface for defining design verification specifications. At the top, there's a toolbar with an 'Update from function' button and a 'Create' checkbox. Below that is a tab bar with 'Specifications' (which is selected, indicated by a red border) and 'Conditions'. The main area is a table with the following columns: Result, Test, Minimum, and Maximum. The table contains 16 rows, each with a checkmark icon in the first column. The 'Test' column lists various timing-related measurements like 'dead\_time', 'up\_clk\_tran:delay', etc. The 'Minimum' and 'Maximum' columns are empty for most rows, except for 'dead\_time' which has 'dead\_band' entered in the 'Test' field.

Result	Test	Minimum	Maximum
✓ dead_time	dead_band		
✓ up_clk_tran:delay	timing		
✓ up_clk_tran:slope	timing		
✓ up_clk_tran:t1	timing		
✓ up_clk_tran:t2	timing		
✓ up_ref_tran:delay	timing		
✓ up_ref_tran:slope	timing		
✓ up_ref_tran:t1	timing		
✓ up_ref_tran:t2	timing		
✓ upb_clk_tran:delay	timing		
✓ upb_clk_tran:slope	timing		
✓ upb_clk_tran:t1	timing		
✓ upb_clk_tran:t2	timing		

To define a specification for design verification, do the following:

1. (Optional) For each row of test and measurement data, type a value in either the *Minimum* or the *Maximum* column, or both.
2. Click *Apply*.

The program creates specifications for those rows of test and measurement data that have a mark in the check box at their far left.

If you leave both *Minimum* and *Maximum* columns empty for a given row, the program does not create a specification for that measurement.

If you leave both *Minimum* and *Maximum* columns empty for all measurements for a given test, the program still runs the test but does not use the results in the spec sheet.

If you disable all measurements for a given test, the program does not run that test.

## **Disabling a Test and Measurement**

To disable a test and measurement, do the following:

1. At the far left of the row for the test and measurement you want to disable, remove the mark from the check box.
2. Click *Apply*.

## **Updating Tests and Measurements from Function Information**

To create the list of tests and measurements for design verification, the program runs a dummy background generation (using only the tests) so that it can take into account the design, function and pin types, and any options when it generates this list. The program creates this list automatically the first time you enable design verification, when you open saved DCM data, or toggle the enable status for design verification (from enabled to disabled and back to enabled again).

If you update function information while using DCM, you need to update the tests and measurements list as described here. The program checks and saves the data before generating. The data must be complete and valid before the program will save it.

To update tests and measurements from function information, do the following:

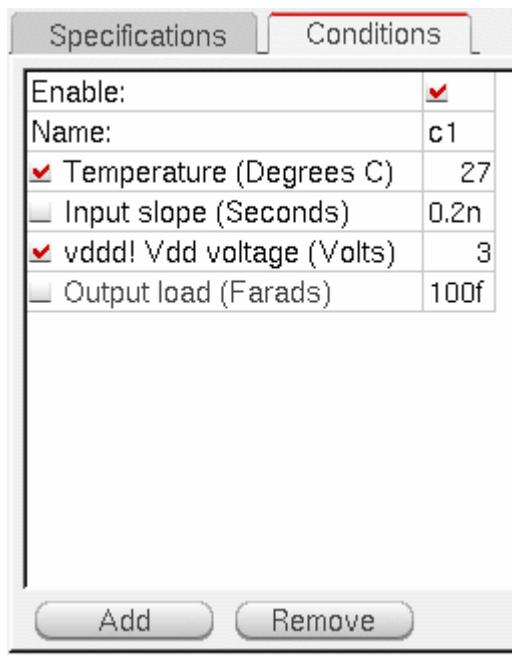
- On the *Design Verification* tab, click *Update from function*.

Updated test and measurement information appears on the *Specifications* tab.

## Specifying Conditions for Design Verification

You can specify conditions for design verification on the *Conditions* tab. The *Conditions* tab is a subtab of the Design Verification tab. Each column in the table defines one condition. On the *Conditions* tab, you can:

- Add any number of columns (conditions) to the table.
- Enable and disable individual condition columns.
- Enable and disable condition parameters (rows in the table).
- Remove a condition column entirely.



See the following topics for more information:

- [Adding Conditions](#) on page 321
- [Enabling Conditions](#) on page 321
- [Disabling Conditions](#) on page 322
- [Renaming Conditions](#) on page 322
- [Removing Conditions](#) on page 322

## Adding Conditions

For each condition you want to add, do the following:

1. On the *Conditions* tab, click *Add*.

A new condition column appears. There is a mark in the *Enable* check box.

If this is the first condition column, the default value for each condition parameter appears in the table cell for that parameter. For each subsequent condition column you add, the values that appear match those in the last column you added.

2. (Optional) To change the name of the condition, click in the *Name* cell and type a new name.
3. For each condition value you want to change, click in the table cell and type a new value.
4. Click *Apply*.

## Enabling Conditions

**Note:** All parameters and conditions are enabled by default.

To enable a condition, do the following:

1. For each condition parameter you want to enable, mark the check box to the left of its name in the table on the *Conditions* tab.  
The program will verify the behavior of the model against the specification of this parameter.
2. For each condition column you want to enable, mark the *Enable* check box at the top of its column.  
The program will use this condition during verification.

## Disabling Conditions

To disable a condition, do the following:

1. For each condition parameter you want to disable, remove the mark from the check box to the left of its name in the table on the *Conditions* tab.

The program will not verify the behavior of the model against the specification of this parameter.

2. For each condition column you want to disable, remove the mark from the *Enable* check box at the top of its column.

The program will not use this condition during verification.

## Renaming Conditions

To rename a condition, do the following:

- On the *Conditions* tab, click in the *Name* cell for that column and type a new name.

The new condition name appears in the *Name* table cell for that column.

## Removing Conditions

For each condition you want to remove, do the following:

1. On the *Conditions* tab, click in any cell in the column for that condition.
2. Click *Remove*.

The program removes the condition column.

---

## Cell Type Development

---

You can create cell types using the dcmDeveloper tool in the Design Characterization and Modeling (DCM) feature of the Virtuoso® Analog Design Environment GXL (ADE GXL).

dcmDeveloper is a cell type development environment that uses OpenDCM, a language embedded within models and test benches. Each cell type is specific to a design function (VCO, OpAmp, etc.) and consists of:

- One .gui file, which defines DCM GUI content (questions posed in the GUI)
- One or more templates -- test or model files with embedded OpenDCM

dcmDeveloper assists with IP migration -- converting hard IP to soft/configurable IP -- and the maintenance and viewing of existing cell types.

To use dcmDeveloper, you load your source data (existing cell type, Spectre netlist, ADE state, schematic, models, etc.). dcmDeveloper includes assistants for substituting pin names with pin types, parameterization, and adding configurability. The tool supports a single, hierarchical tree view of the test bench. You can specify features, options, etc. for the GUI setup, then check your data and save into a directory for DCM.

You can also use the [dcmDeveloper wizard](#) to create a cell type. For more information, see [Understanding the dcmDeveloper Wizard](#).

For more information the OpenDCM template syntax, see [Appendix A, "OpenDCM."](#)

See the following topics for more information:

- [Launching dcmDeveloper to Create New Cell Types](#)
- [Adding and Configuring Test Benches](#)
- [Loading State Files](#)
- [Adding Models](#)
- [Specifying Cell Type Generation Options](#)
- [Specifying the Output Destination](#)

## Virtuoso Analog Design Environment GXL User Guide

### Cell Type Development

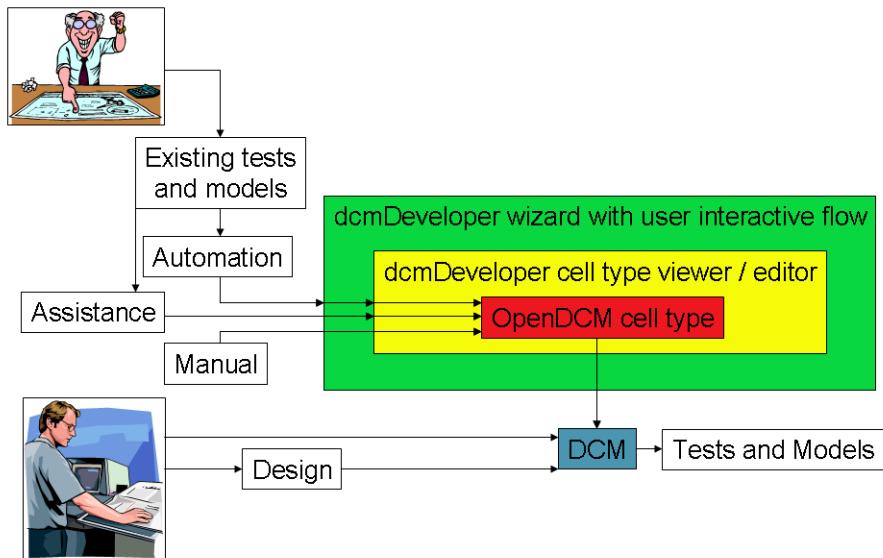
---

- [Saving the dcmDeveloper Data](#)
- [Closing dcmDeveloper](#)
- [Viewing an Existing Cell Type](#)

**Note:** See [Chapter 7, “Model Generation”](#) for information about DCM’s model generation capabilities.

## Understanding the dcmDeveloper Wizard

The dcmDeveloper wizard provides an interactive flow for the migration of existing tests and models used to create new cell types for the support of new design types. You can also use the wizard to add new tests and models to an existing cell type.



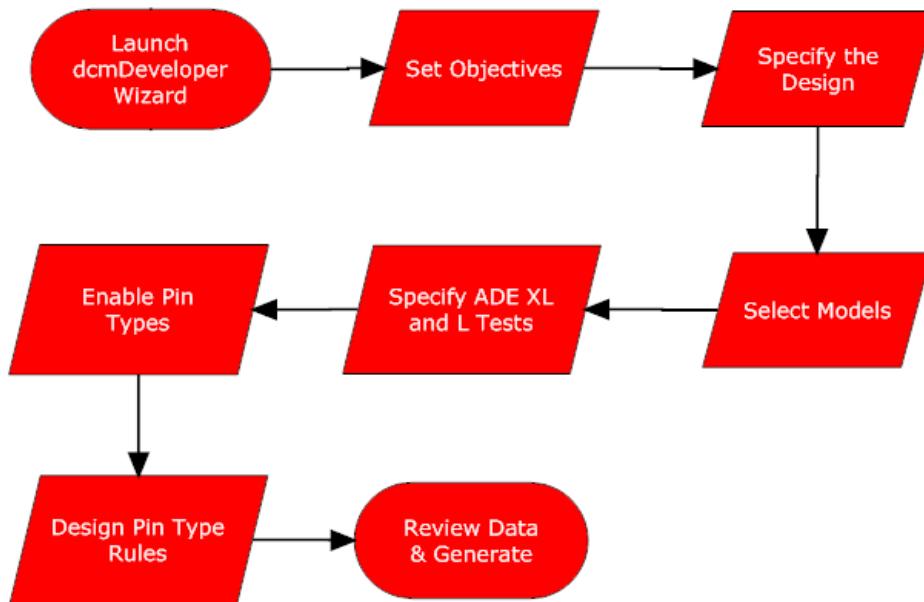
You begin by stating your objectives (features and functionality) for the new cell type. The stated objectives determine the flow through the wizard and the data requested of you by the wizard.

When the wizard finishes processing your cell type, you can interact further with it in dcmDeveloper.

The following sections outline the flows for bottom-up and top-down modeling and design verification. Note that you can select more than one objective type, in which case you will follow the flow for all specified objectives.

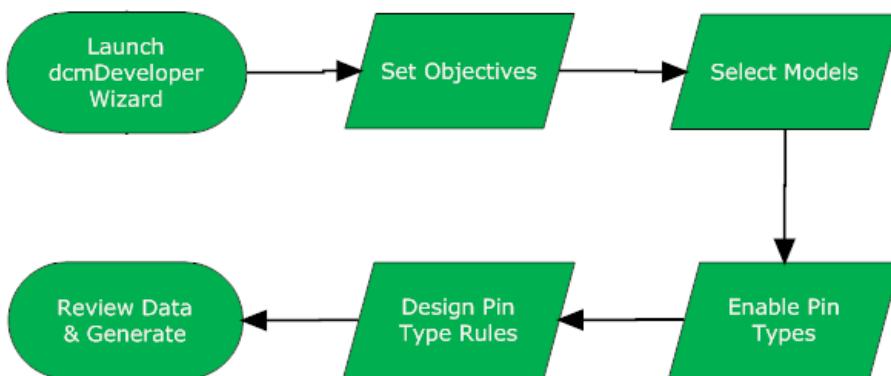
## Bottom-Up Modeling

The process flow in the Wizard for bottom-up modeling is as follows:



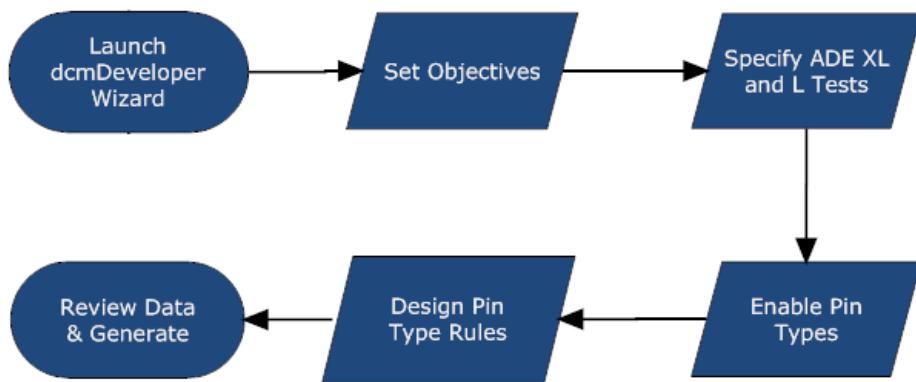
## Top-Down Modeling

The process flow in the Wizard for top-down modeling is as follows:



## Design Verification

The process flow in the Wizard for design verification is as follows:



## **Launching dcmDeveloper to Create New Cell Types**

You can create new cell types from existing hard IP using dcmDeveloper.

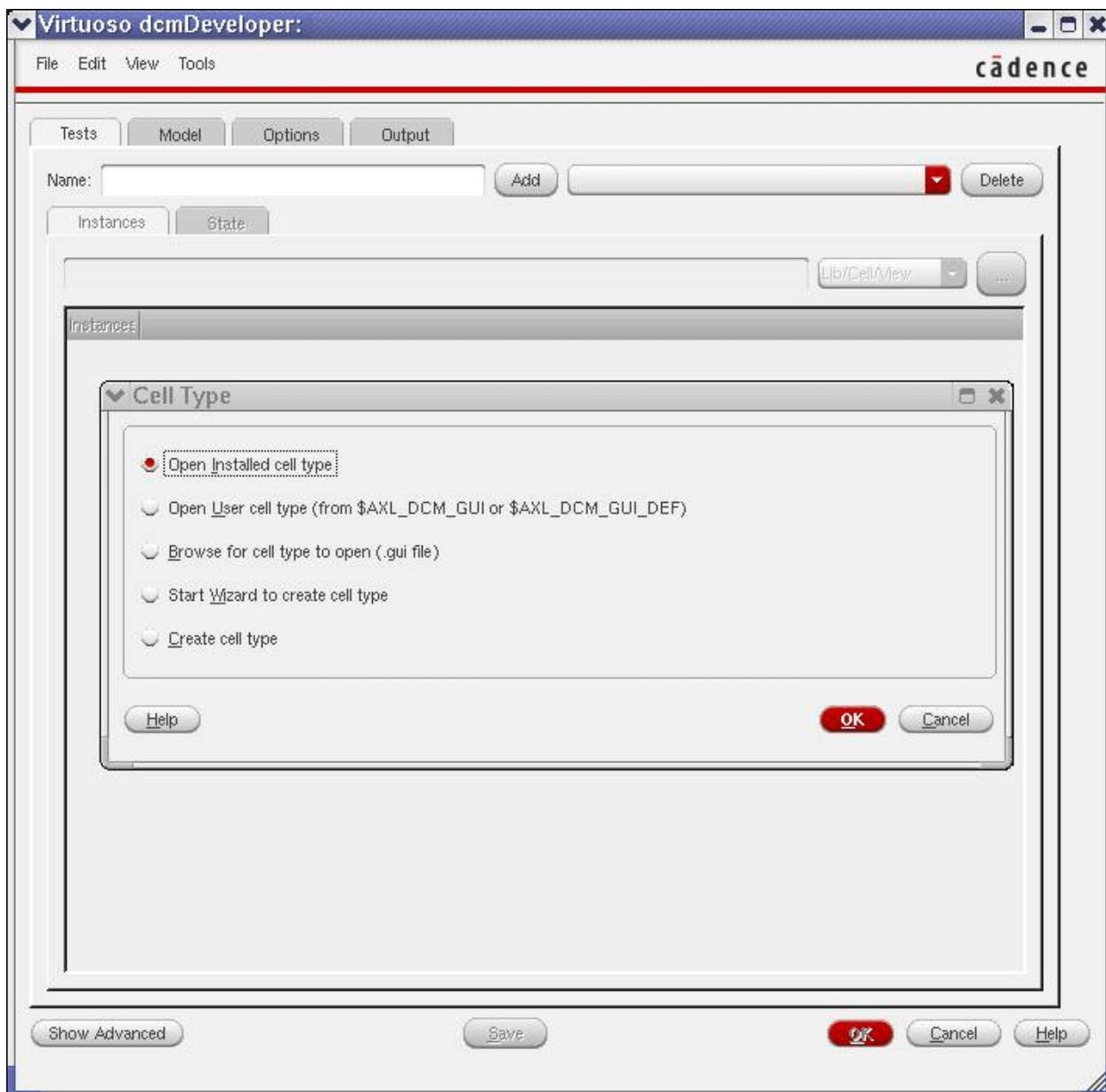
To launch dcmDeveloper, do one of the following:

- From the CIW, choose *Tools — Characterization and Modeling — Cell Type Development*.
- From the main DCM interface, choose *Tools — Create new function*.

## Virtuoso Analog Design Environment GXL User Guide

### Cell Type Development

The dcmDeveloper window appears, along with the Cell Type form.



### Understanding the dcmDeveloper Form

The Tests page enables you to add and select tests. Within the Tests page, you can add components and measurements on the Instances page. Within this page, you can browse and load new data. Nested OpenDCM macro commands are displayed in the list box, and you can double-click an instance name to open an instance editor. Slick *Show Advanced* to view

options for modifying instances and pins and adding measures. On the State tab, you can enable, disable, and parameterize individual state files and their contents.

You can view the resulting OpenDCM data for the different languages on the Models page. Click the Options tab to specify options, and click the Output tab to specify the resulting output from dcmDeveloper.

### **Utilizing Advanced Options**

At any time, you can click *Show Advanced* to add or modify instances, measures, pin types, etc. Many functions are also available through right-click context menus.

You can now:

- [Launch the dcmDeveloper Wizard](#) to create a new cell type
- [Add and Configure test benches](#) (not required for top-down models)
- [Add Models](#) (not required for design verification)
- [Specify Cell Type Generation Options](#)
- [Specify the Output Destination and Options](#)

## Creating a New Cell Type in the dcmDeveloper Wizard

To launch the dcmDeveloper wizard, use one of the following methods:

- In the main dcmDeveloper window, choose *File — Wizard*.
- In the Cell Type form, select the *Start Wizard to create cell type* radio button, then click *OK*.

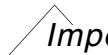
The Cell Type Development Wizard appears.



You can now set your objectives in the Objectives tab.

### Setting Objectives

In the Objectives tab, you define which model types (Bottom-Up and/or Top-Down), model formats, and design verification will be supported by the new cell type. The objectives you choose on this page determine the flow of the wizard. For example, if you choose Bottom-Up or Design Verification, the wizard will request design and test bench information. If you choose Bottom-Up or Top-Down, the wizard will request model files.

 *Important*

Currently, the Wizard is most comprehensive in supporting Verilog-A[MS] through automatic template creation and assistance with Calibration.

**1. To specify Bottom-Up Modeling:**

- a.** Select the *Bottom-Up Modeling* check box.
- b.** Select one of the following behavioral models:
  - Verilog-A[MS]
  - Verilog-D
  - Liberty (.lib)

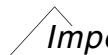
**2. To specify Top-Down Modeling, select the *Top-Down Modeling* check box.**

The Verilog-A[MS] check box is automatically selected.

**3. To specify design verification, select the *Design Verification* check box.**

**4. Under Options, select the *Create configurable tests and/or models* check box if you want to be able to create pin type rules.**

If this option is disabled, the Pin Type Rules tab is disabled and the rule for all pin types is set to ‘==1’, which means that each pin type is to be related to just one design pin.

 *Important*

If configurable tests and models are to be created, DCM\_IF (condition) and/or DCM\_GENERATION (loop/repeat) macros must be added around instances or model text when the pin types are used. Pin types that support less than one design pin (optional usage) should use DCM\_IF. Pin types that support one or more design pins should use DCM\_CONDITION.

**5. Select the *DCM created power supply sources* check box if you want DCM to create supply sources for any design pins that require them.**

Designs with different numbers of supply pins can thus be supported by the cell type. If this option is enabled, the wizard removes supply sources within test benches and instead utilizes the default DCM supply pin types (Vdd and Vss).

If this option is disabled, the wizard assumes that the test benches provide supply sources.

**6. Click *Next*.**

If you specified Bottom-Up Modeling or Design Verification in your objectives, you will next specify your design information in the Design tab.

If you specified Top-Down Modeling, you will next specify your models in the Model tab.

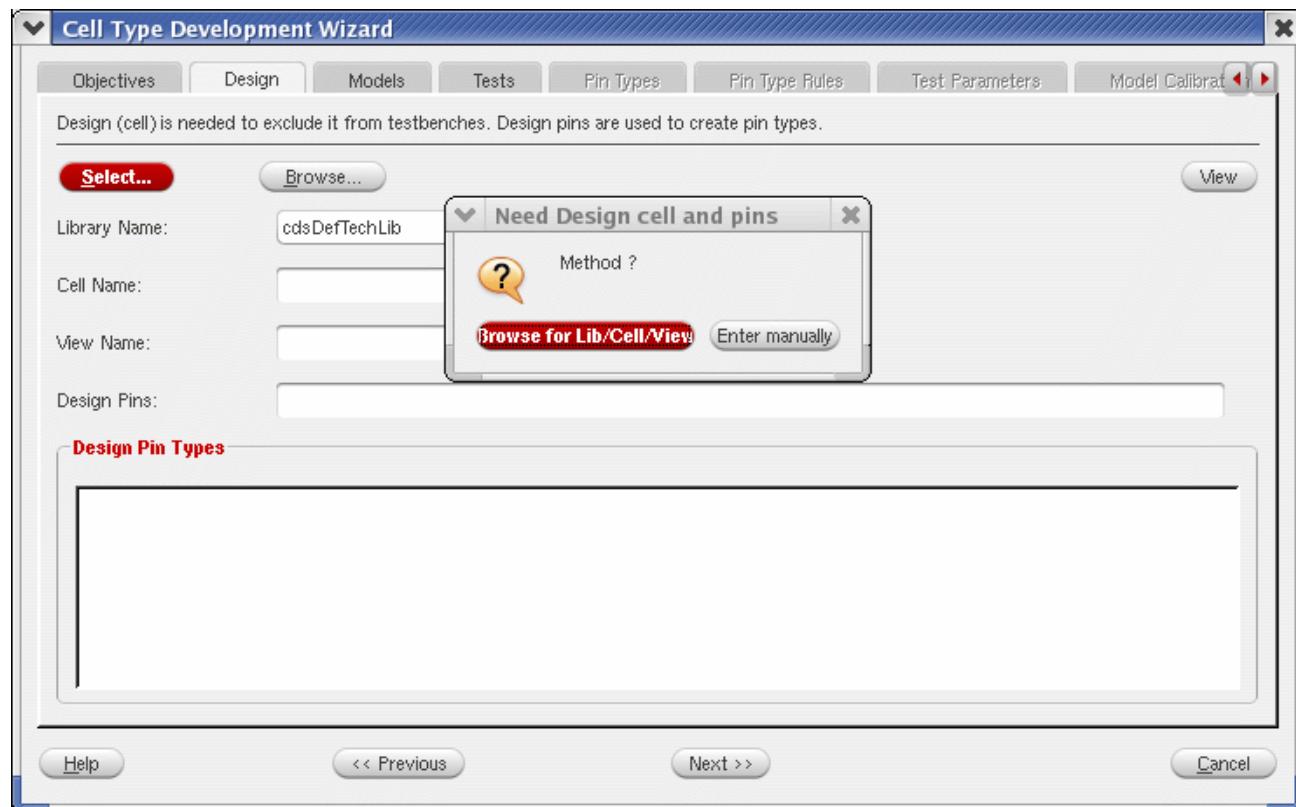
## Specifying Design Information

You specify the design in the Design tab. The wizard requires information about the design to map the design pins to nets used within the test benches, to identify the design instance, and to identify the Vdd and Vss pins of the design if the *DCM created power supply sources* option was enabled in the cell type objectives.

To specify design information.

1. If necessary, select the Design tab.

The Design page appears, along with the Need Design cell and pins form.



2. Specify the design:

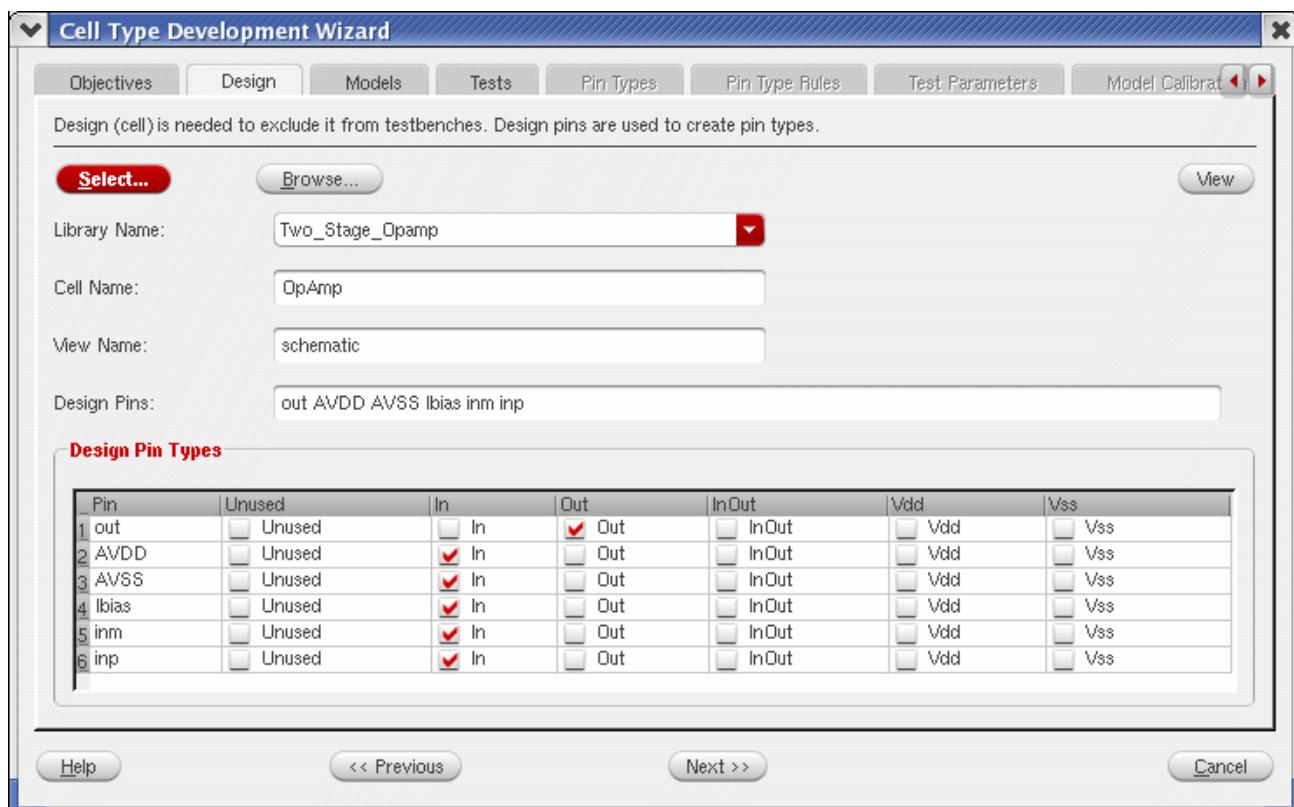
- ❑ To browse for the design, click *Browse for Lib/Cell/View* in the Need Design cell and pins form, or click *Browse* in the Design page.

## Virtuoso Analog Design Environment GXL User Guide

### Cell Type Development

- To specify the design name, click *Enter manually* in the Need Design cell and pins form, then enter the cell name in the Cell Name field on the Design page. Press *[Enter]* when finished.
- To select an open schematic, click *Select* in the Design page, then select a design instance on a schematic.

After the design is loaded, the wizard extracts the design pins and uses the information to populate the Design Pin Types table, then determines the pin directions from the schematic (if available).



The Design Pin Types table identifies the Vdd and Vss pins (required if the *DCM created power supply sources* option was enabled in the cell type objectives), which the wizard uses to determine the nets connected to the design supply pins and thus the instances that provide the supply voltages. The wizard then removes the supply sources from the test template so that the DCM-generated sources can be added.

3. If you selected the *DCM created power supply sources* option in the Objectives, select the check boxes for the Vdd and Vss power pins in the Design Pin Types table.

Design Pin Types							
Pin	Unused	In	Out	InOut	Vdd	Vss	
1 out	<input type="checkbox"/> Unused	<input type="checkbox"/> In	<input checked="" type="checkbox"/> Out	<input type="checkbox"/> InOut	<input type="checkbox"/> Vdd	<input type="checkbox"/> Vss	
2 AVDD	<input type="checkbox"/> Unused	<input type="checkbox"/> In	<input type="checkbox"/> Out	<input type="checkbox"/> InOut	<input checked="" type="checkbox"/> Vdd	<input type="checkbox"/> Vss	
3 AVSS	<input type="checkbox"/> Unused	<input type="checkbox"/> In	<input type="checkbox"/> Out	<input type="checkbox"/> InOut	<input type="checkbox"/> Vdd	<input checked="" type="checkbox"/> Vss	
4 Ibias	<input type="checkbox"/> Unused	<input checked="" type="checkbox"/> In	<input type="checkbox"/> Out	<input type="checkbox"/> InOut	<input type="checkbox"/> Vdd	<input type="checkbox"/> Vss	
5 inm	<input type="checkbox"/> Unused	<input checked="" type="checkbox"/> In	<input type="checkbox"/> Out	<input type="checkbox"/> InOut	<input type="checkbox"/> Vdd	<input type="checkbox"/> Vss	
6 inp	<input type="checkbox"/> Unused	<input checked="" type="checkbox"/> In	<input type="checkbox"/> Out	<input type="checkbox"/> InOut	<input type="checkbox"/> Vdd	<input type="checkbox"/> Vss	

4. Click *Next*.

If you specified Bottom-Up Modeling, you will next specify your models in the Model tab.

If you specified Design Verification, you will next specify your tests.

## Viewing the Design

You can view the design you have selected:

- Click *View*.

The cell view specified for the design appears.

## Selecting the Models

In the Models tab, you specify a model for each enabled Bottom-Up and/or Top-Down format.

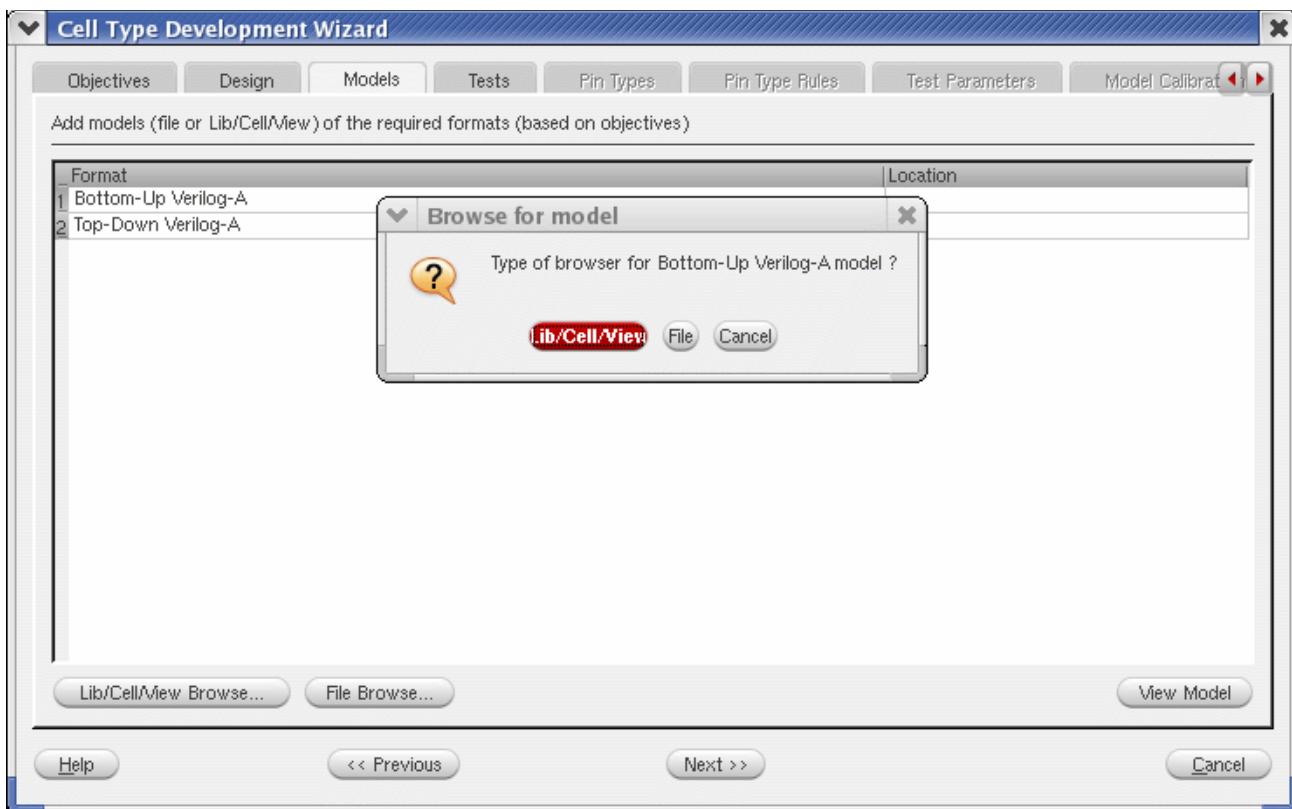
To select models:

1. If necessary, select the Models tab.

## Virtuoso Analog Design Environment GXL User Guide

### Cell Type Development

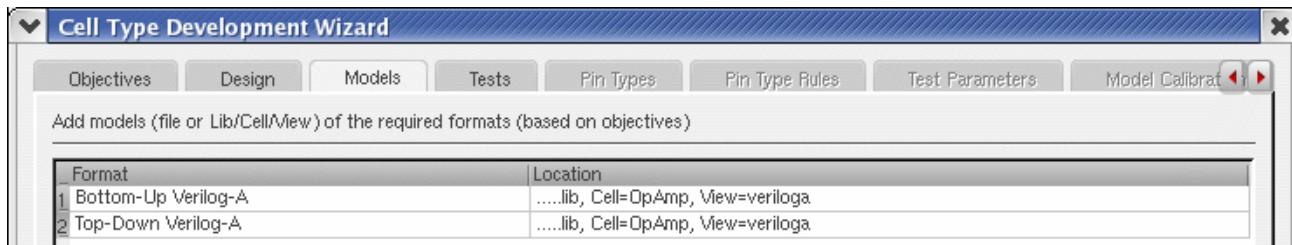
The Models page appears, along with the Browse for model form.



#### 2. Specify the model:

- ❑ To browse for the model, click *Browse for Lib/Cell/View* in the Browse for model form, or click *Lib/Cell/View Browse* on the Model page.
- ❑ To load a file, click *File* in the Browse for model form, or click *File Browse* in the Model page.

The Wizard loads and analyzes the model, then creates the template and pin types.



#### 3. Click *Next*.

If you specified Bottom-Up Modeling, you will next specify tests.

If you specified Top-Down Modeling, you will next enable pin types.

## **Viewing the Model**

You can view the model you have selected:

1. Select the model in the Models table.
2. Click *View Model*.

The cell view for the model appears.

## **Specifying Tests**

Next, you specify tests. You can load all tests within an ADE XL view. When the XL view is loaded, you can also choose any ADE XL and ADE L variables you want to load in addition to the tests. You can also choose to add an ADE L test by specifying the test name and browsing for the schematic and state.

You can load multiple ADE XL views and additional ADE L tests.

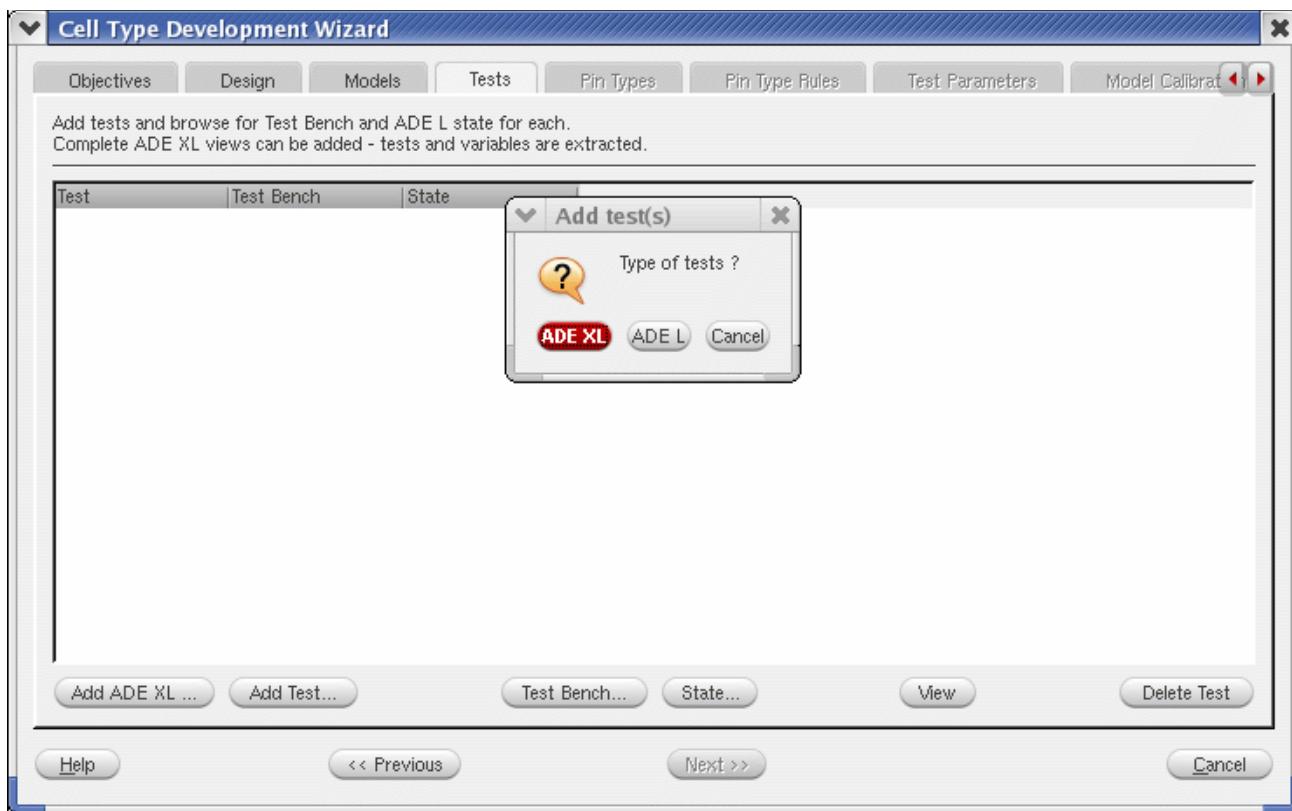
To begin specifying tests:

- Select the Tests tab if necessary.

# Virtuoso Analog Design Environment GXL User Guide

## Cell Type Development

The Tests page appears, along with the Add test(s) form.



You can now:

- specify ADE XL Tests
- specify ADE L Tests

### ADE XL Tests

1. To browse for the ADE XL view, click *ADE XL* in the Add test(s) form, or click *Add ADE XL* on the Tests page.

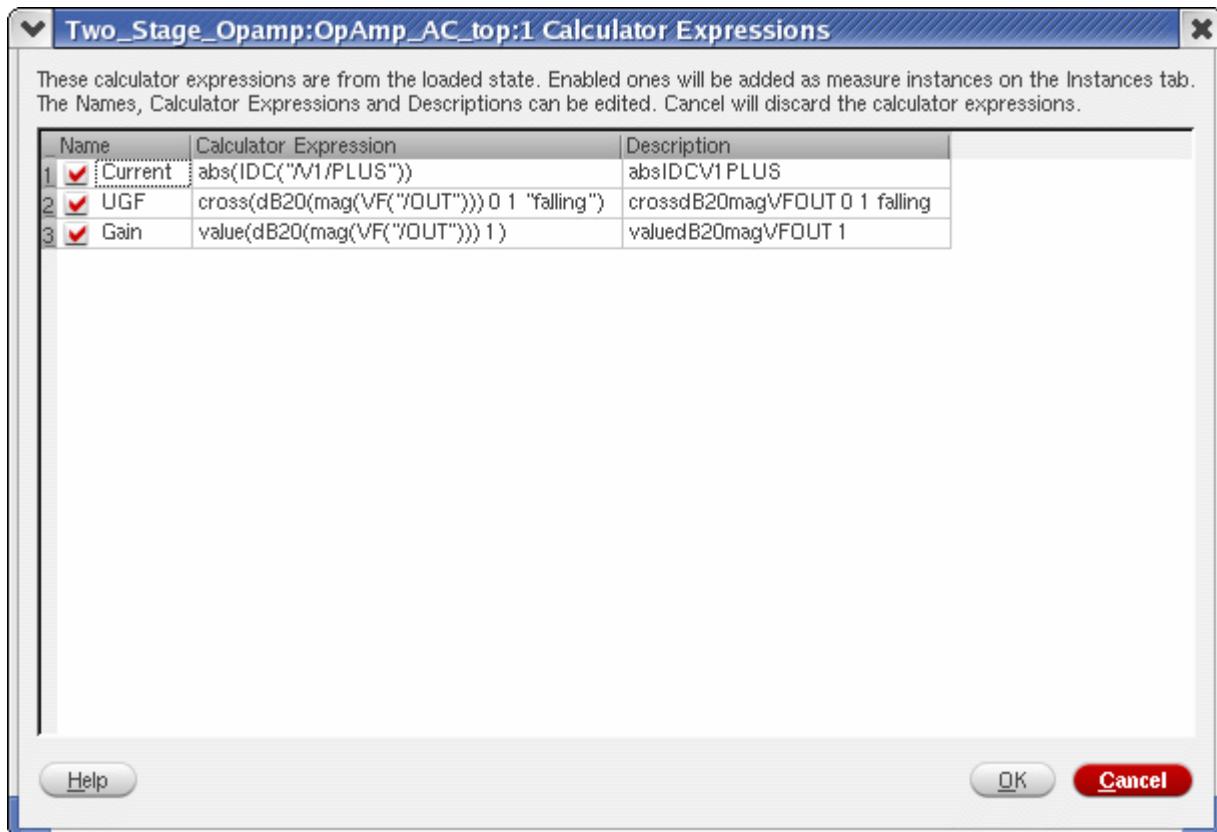
The Select ADE XL view to load form appears.

2. Browse to the ADE XL view. Note that the view type must be *adexl*.

## Virtuoso Analog Design Environment GXL User Guide

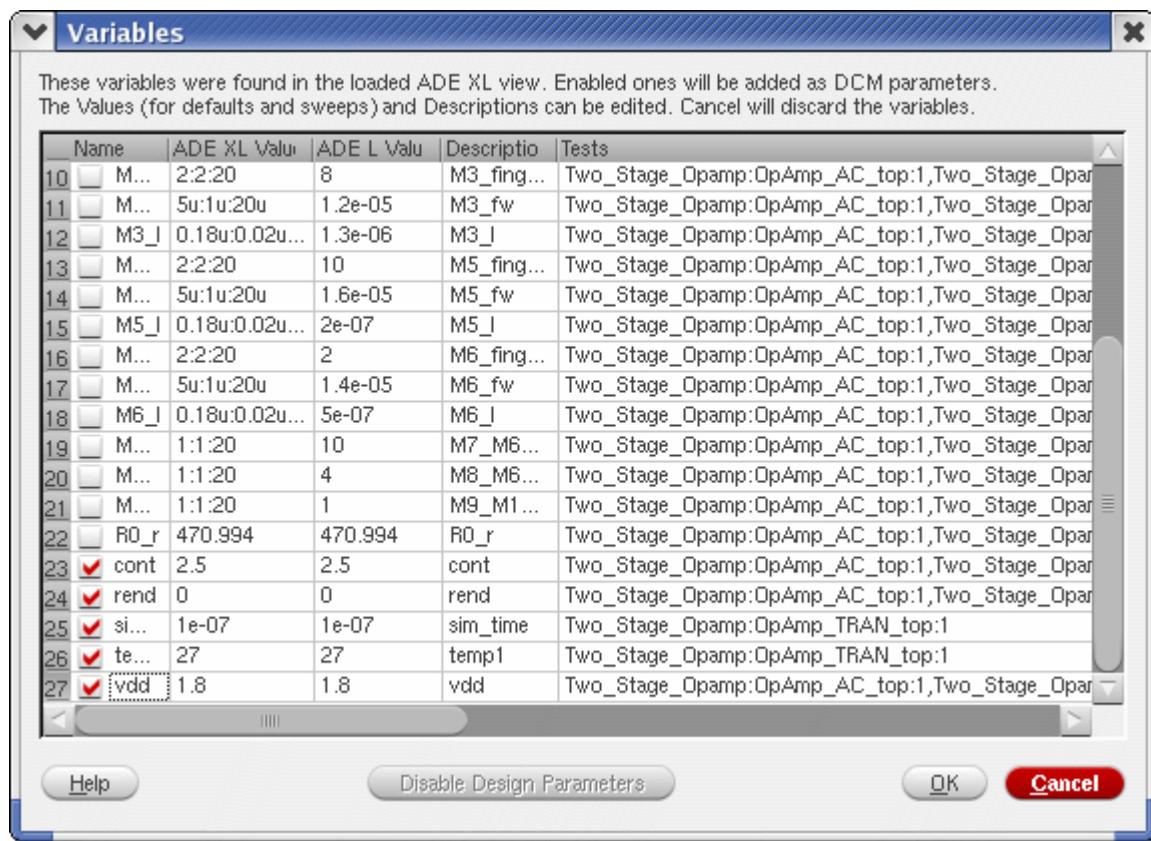
### Cell Type Development

The calculator expressions from one of the test benches attached to the specified state are extracted and displayed in a Calculator Expressions form.

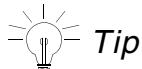


3. All calculator expressions are selected by default. Disable the check boxes for expressions you do not want added.
4. Modify the expression names, expressions, and descriptions as desired.
5. Click *OK* in the Calculator Expressions form.
6. The calculator expressions for any additional test benches are displayed. Repeat step 3 through step 5 for all test benches.

The Variables form appears, listing all of the variables from the specified state. The variables used within testbenches are automatically selected, while all design variables are automatically disabled.



7. Enable the check boxes for variables you want added as DCM parameters, and disable the check boxes for variables you do not want added.



*Tip*  
The default enabled/disabled variables are recommended. A variable that is specific to the design should not normally be enabled as the cell type must work with different designs.

8. Modify the variable values and descriptions as desired.
9. Click *OK* in the Variables form.



Clicking *Cancel* will discard all variables.

The test information is displayed in the Tests tab of the Wizard.

## ADE L Tests

1. To add an ADE L Test, click *ADE L* in the Add test(s) form, or click *Add Test* in the Model page.

The New Test form appears.



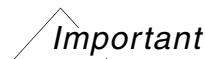
2. Enter a test name in the Test Name field and click *OK*.

The Browse for test bench form appears.



3. Specify your test bench:

- To browse for a lib/cell/view, click *Lib/Cell/View*.
- To browse for a spectre netlist, click *Spectre Netlist*.



The selected testbench Lib/Cell/View must be a schematic.

The Browse for ADE L state form appears.



**4. Specify your ADE L state:**

- To browse for an ADE state using an ADE L state browser, click *State*.
- To browse for a directory, click *State Directory*.

The calculator expressions from the test benches attached to the specified state are extracted and displayed in a Calculator Expressions form.

- 5. Enable the check boxes for calculator expressions you want added as measure instances, and disable the check boxes for expressions you do not want added.**
- 6. Modify the expression names, expressions, and descriptions as desired.**
- 7. Click *OK* in the Calculator Expressions form.**

The state and test bench information are displayed with the test name in the Tests page of the Wizard.

### **Selecting a New Test Bench for a Test**

To change the test bench specified for a test:

- 1. Select the test name in the Tests table.**
- 2. Click *Test Bench*.**

The Browse for test bench form appears.

- 3. Specify your test bench:**
- To browse for a lib/cell/view, click *Lib/Cell/View*.
  - To browse for a spectre netlist, click *Spectre Netlist*.



The selected testbench Lib/Cell/View must be a schematic.

### **Selecting a New State for a Test**

To change the state specified for a test:

- 1. Select the test name in the Tests table.**
- 2. Click *State*.**

The Browse for ADE L state form appears.

3. Repeat the procedure for selecting a state as specified in [step 4 through step 7 of ADE L Tests](#).

### **Deleting a Test**

To delete a test:

1. In the Tests table, select the test name you want to delete.
2. Click *Delete*.

### **Viewing a Schematic**

You can view the schematic for any test bench.

1. In the Tests table, select the test bench name you want to view.
2. Click *View*.

### **Dismissing the Tests Page**

When you are finished specifying all tests, click *Next*.

You will now [enable pin types](#).

### **Enable Pin Types**

Pin types provide a design-neutral way of defining test benches and models. As the objective is to support other designs, the pin names for the current design and/or models must be replaced with a pin type. These pin types can then be associated with pin names for a different design when the cell type is used.



*Caution*

***You cannot add new pin types in the Wizard, only enable existing pin types. You can create new pin types in dcmDeveloper. See [Adding and Modifying Pin Types](#) for more information.***

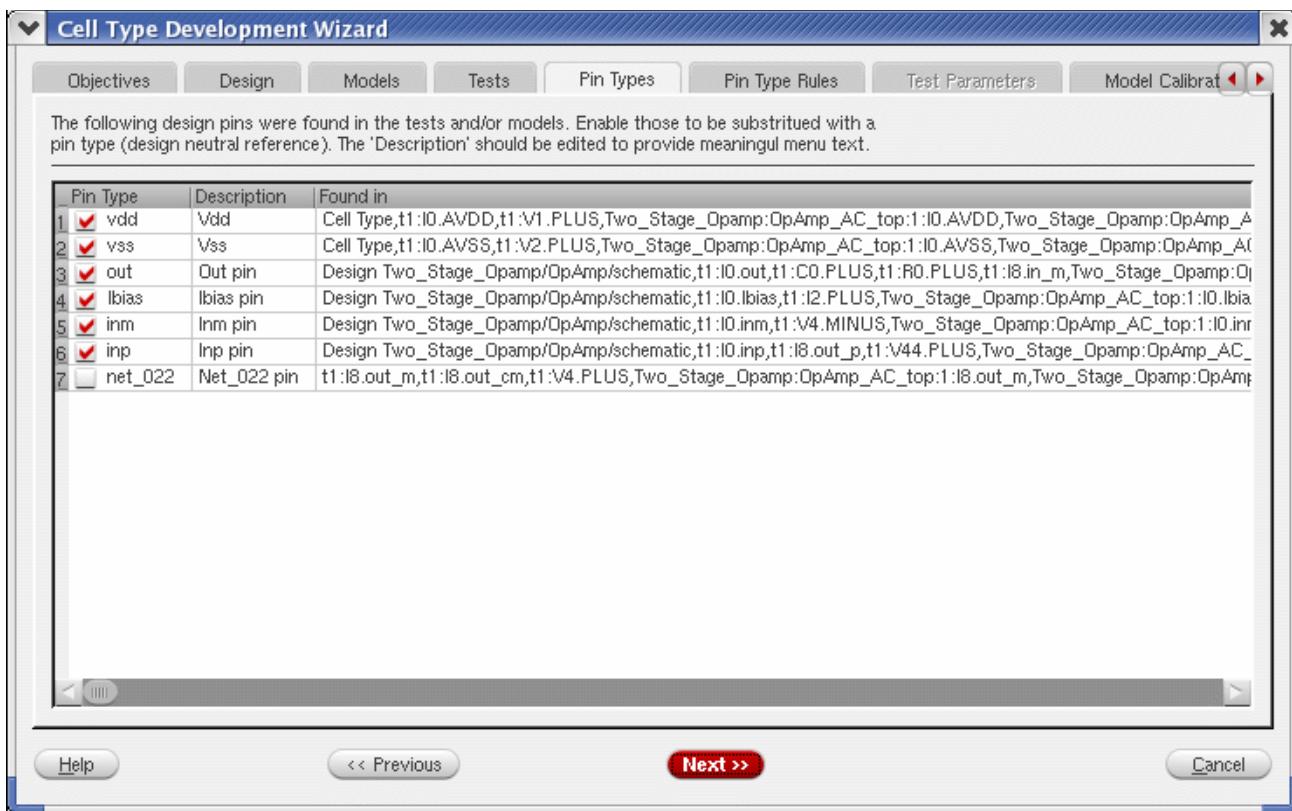
To specify your pin types:

1. If necessary, select the Pin Types tab.

## Virtuoso Analog Design Environment GXL User Guide

### Cell Type Development

The Pin Types page appears.



The table on this page includes a list of design pin names, model pin names, and nets found in specified tests that are not connected to design pins. The design and model pin types are selected by default, as are the Vdd/Vss pin types if the *DCM created power supply sources* option is enabled on the Objectives tab.

This table also includes a list of the locations in which the pin type was found. Note that the design instance within the test bench is analyzed and any instances connected to a net connected to a design pin are included. For example, if design pin *A* is connected to net *myNetA* which is connected to *V1 plus*, then a pin type *A* would be created and the "Found in" description would specify "*V1.plus*".

2. In the Pins table, enable and disable pins for pin type replacement.
3. Modify the default description for pins as desired.
4. Click *Next*.

You will now define pin type rules.

## Define Pin Type Rules

Pin type rules define how pin types can be used in new designs. The easiest method is to allow each pin type to be related to just one design pin, which means that the design must contain exactly one pin of the type in order to use the pin type. However, if there are no pins, or more than one pin, the type cannot be used.



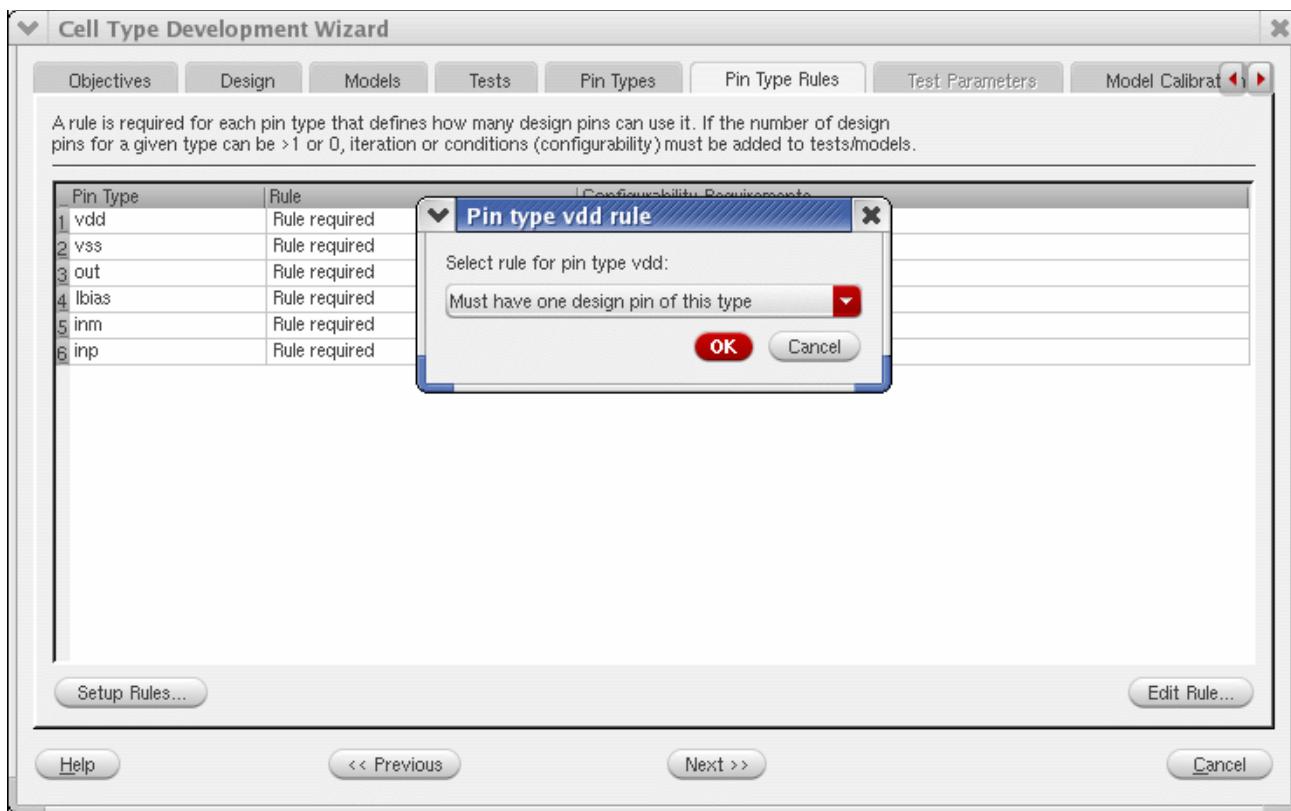
The Pin Type Rules page is only displayed when the Create configurable tests and/or models check box is selected on the Objectives page. If that option is not selected, the rule for all pin types is set to ‘==1’, which means that each pin type is to be related to just one design pin.

Cell types can also support optional pin types (where there can be 0 or 1 design pins of this pin type) and pin types that support multiple design pins (where there can be 0 or more, or 1 or more, design pins of this pin type). If these rules are used, additional work is required in dcmDeveloper to define the parts of the test and model that should be included / excluded if there is, or is not, a pin of this type (for optional pin types) or to define the parts of the test and model that should be repeated if there are multiple design pins of this type.

You must define a rule for each pin type. When a design is used with this cell type, the rules are checked and if broken, error messages are issued and therefore tests and models will not be created.

- If necessary, select the Pin Type Rules tab.

The Pin Type Rules page appears, along with the Pin type rule form.



The table on this page includes the list of pin types defined in the Pin Types page. You must define a rule for each pin type.

You can either:

- [Define a rule for each pin type.](#)
- [Define one rule for all pin types.](#)

### Defining a Rule for Each Pin Type

You can define a rule per pin type. When you first select the Pin Type Rules page, the Pin type rule form appears for the first pin type listed.

To define a rule for this pin:

1. If necessary, click *Define Rules* to view the Pin type rule form.
2. Select one of the following rules from the pull-down list.
  - Must have one design pin of this type*

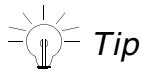
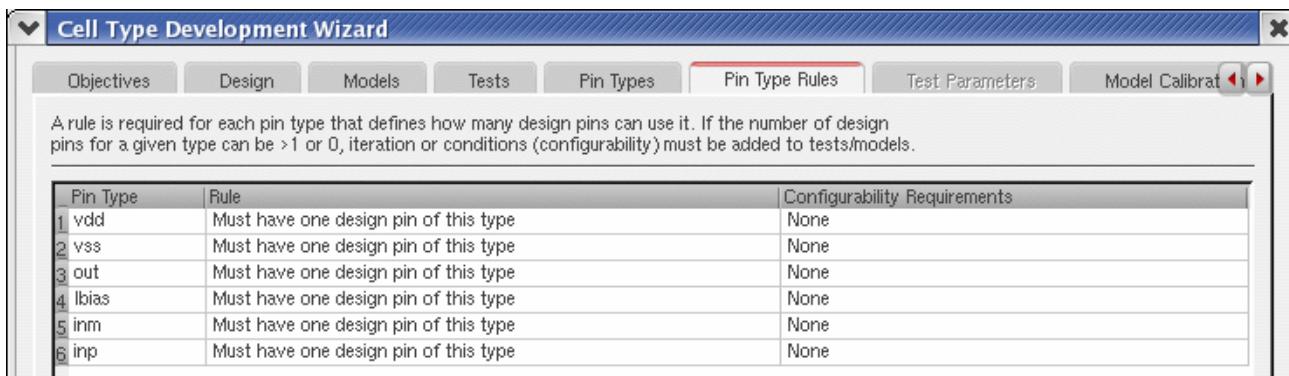
- Design pin is optional for this type*
- Multiple (1..n) design pins of this type*
- Optional Multiple (0..n) design pins of this type*

**3.** Click *OK*.

The Pin type rule form appears for the next pin type listed.

**4.** Repeat step 2 and step 3 for all pin types.

When you are finished, the Wizard populates the Rule fields in the table with the specified rule for each pin.



**Tip**  
The Configurability column in the table contains hints on what needs to be done within the test and/or model templates when the pin type is used for the selected pin type rule.

**5.** Click *Next* when you are done specifying pin type rules.

If you specified Bottom-Up Modeling, you will next specify model calibration.

Otherwise, you will now review your data.

### Defining a Rule for All Pin Types

You can also define one rule for all pin types:

1. If necessary, click *Cancel* to dismiss the Pin type rule form.
2. Right-click in the Rules table and choose one of the *Set to...* options.
3. Click *Next* when you are done specifying pin type rules.

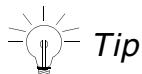
If you specified Bottom-Up Modeling, you will next specify model calibration. Otherwise, you will now review your data.

## Modifying a Rule

Once you have set up a rule, you can modify it:

1. In the Rules table, select the rule you want to modify.
2. Click *Edit Rule*.

The Pin type rule form for that pin appears.



You can also double-click the rule in the Rules table.

3. Select a rule from the pull-down list.
4. Click *OK*.

The Rule field for the pin is updated with the new rule.

## Calibrating the Verilog-A[MS] Model

The Calibration page displays the parameters and variables from the Verilog-A[MS] model and the results from the tests. In order to calibrate the generated model to the measured behavior of the design, you must correlate the measured result from a test to a parameter or variable of type "real" within the model.

When the cell type is created, DCM uses this model calibration information to process the model template, then automatically insert the DCM\_CALIBRATE macro calls that refer to the test and result. For model variables, DCM replaces value assignments with the DCM\_CALIBRATE call. For model parameters, DCM comments out the parameter declaration, adds a real variable declaration, and adds a variable assignment to the analog block of the model.

When the cell type model is used with a new design, the DCM\_CALIBRATE statements are replaced with a \$table\_model call that refers to a table file created from the test and result measurements swept over the parameters specified by the cell type user.

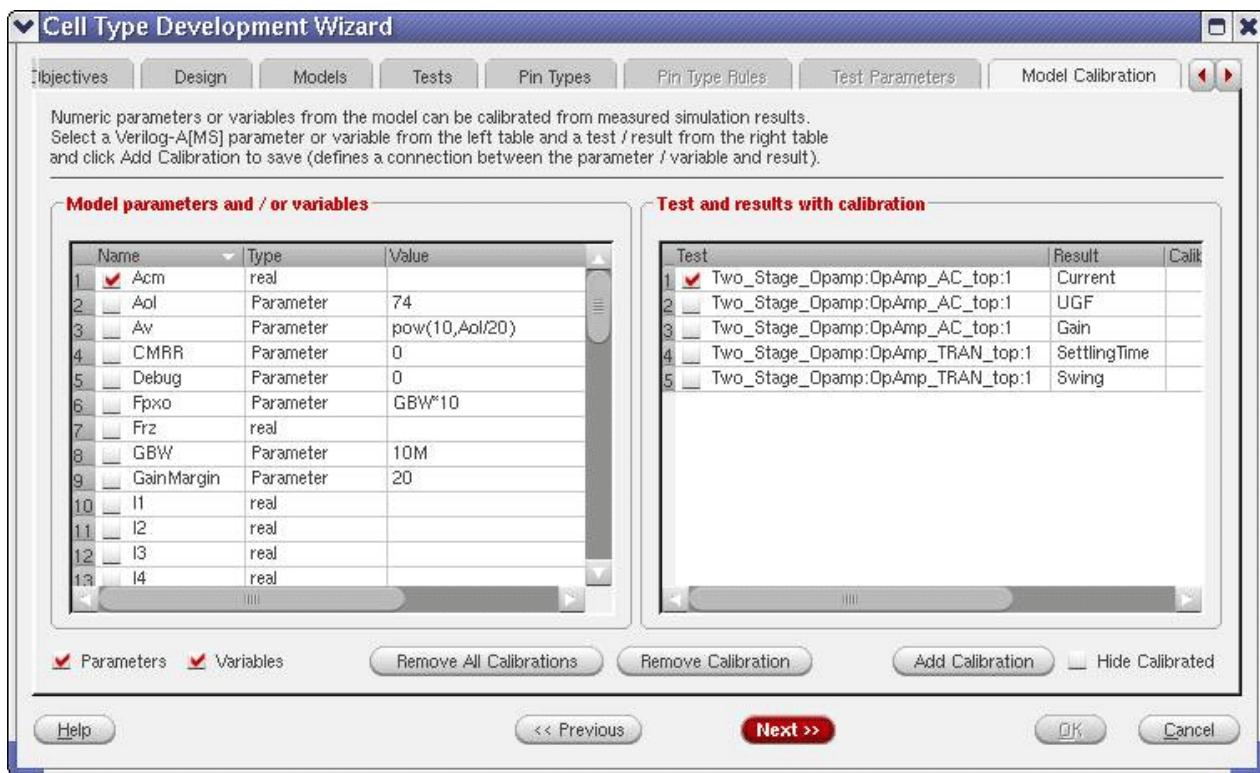
To define an association between a model parameter or variables and a test result:

1. If necessary, select the Model Calibration tab.

# Virtuoso Analog Design Environment GXL User Guide

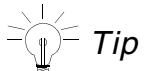
## Cell Type Development

The Model Calibration page appears.



The tables on this page list model parameters and variables and test results.

2. Select the check box for a parameter or variable name in the Model parameters and/or variables table.



You can change the view of this table by selecting and de-selecting the *Variables* and *Parameters* check boxes at the bottom.

3. Select a test result in the Test and results with calibration table.
4. Click *Add Calibration*.

The connection is defined and the correlated variable or model parameter is then listed in the Calibration field in the Test and results table.

5. Repeat step 2 through step 4 for all connections you want to define.
6. When you are finished, click *Next*.

You will now review your data.

## Hiding Calibrated Values

To hide the model parameters/variables and test results that have already been defined for model calibration:

- Select the *Hide Calibrated* check box.

## Removing a Calibrated Value

To remove a calibration:

1. Select the calibrated test name in the Test and results with calibration table.
2. Click *Remove Calibration*.

## Removing All Calibrated Values

To remove all calibration associations that have been defined:

- Click *Remove All Calibrations*.

## Reviewing the Cell Type

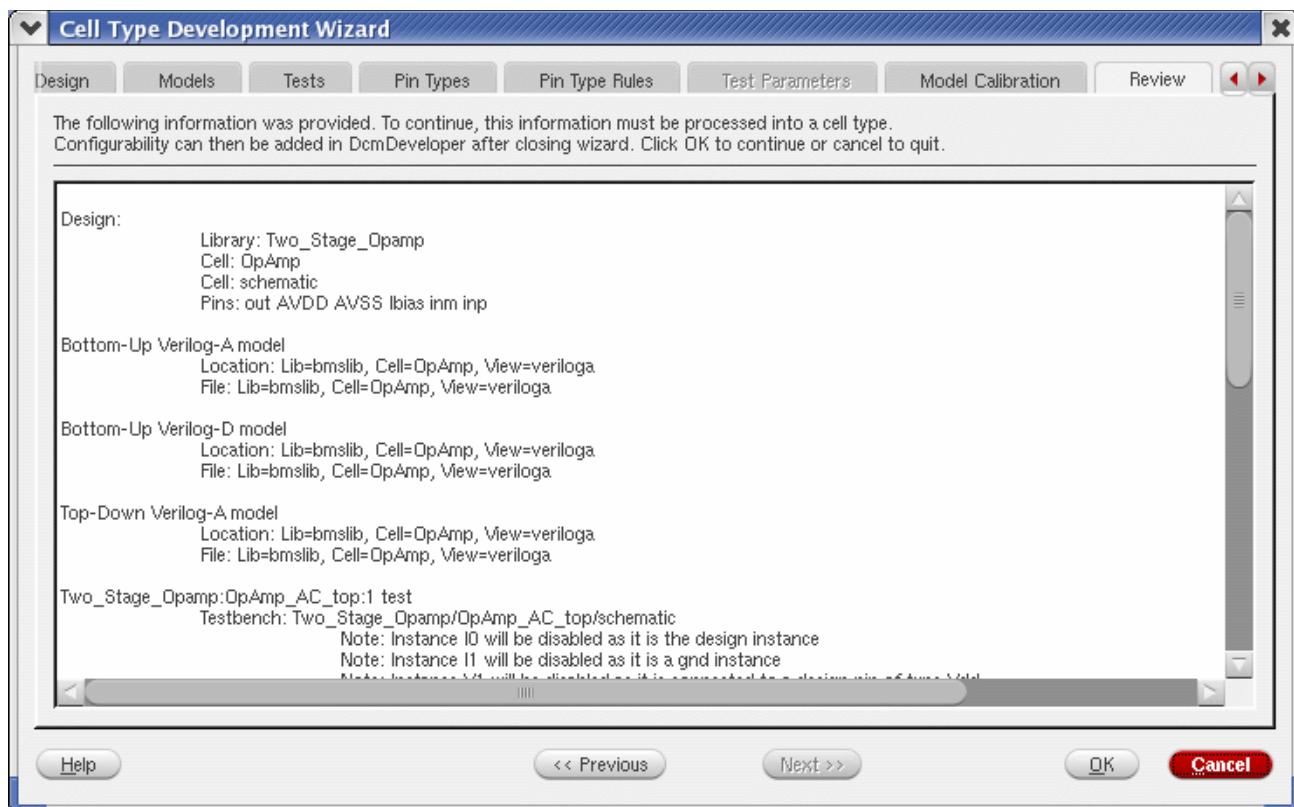
When you are finished adding data in the dcmDeveloper Wizard, you can review the data.

1. If necessary, select the Review tab.

# Virtuoso Analog Design Environment GXL User Guide

## Cell Type Development

The Review page appears.



2. Review the data provided. If necessary, select previous tabs to modify the information.
3. When finished, click *OK* to generate the cell type.

A dialog appears, listing steps that have been taken to generate the cell type.



Although the cell type has been generated, you can return to the wizard to make minor modifications then re-generate, or you can close the wizard.

4. Click *Yes* to close the wizard, or click *No* to return to the wizard.

If you clicked *Yes*, the Tasks form appears.

If you clicked *No*, the Wizard reappears.

## Completing Tasks

The dcmDeveloper Wizard task form contains a list of the tasks that need to be carried out in the main dcmDeveloper window in order to complete development of the cell type.

Each task is associated with a tab or dialog in DcmDeveloper. As the task is selected, the Wizard displays the relevant tab or dialog required to carry out the task. If forms are displayed, these must be dismissed before moving on to the next task.

There are a number of tasks that may be required in addition to those shown:

- If Verilog-D or Liberty .lib models were loaded, these must be modified on the relevant model tab to create the template.
- If a configurable cell type is being created and at least one pin type has a rule that is not ‘== 1 design pin’, the required DCM\_IF and/or DCM\_GENERATE macros must be added around the use of those pin types within the test instances or model text lines.

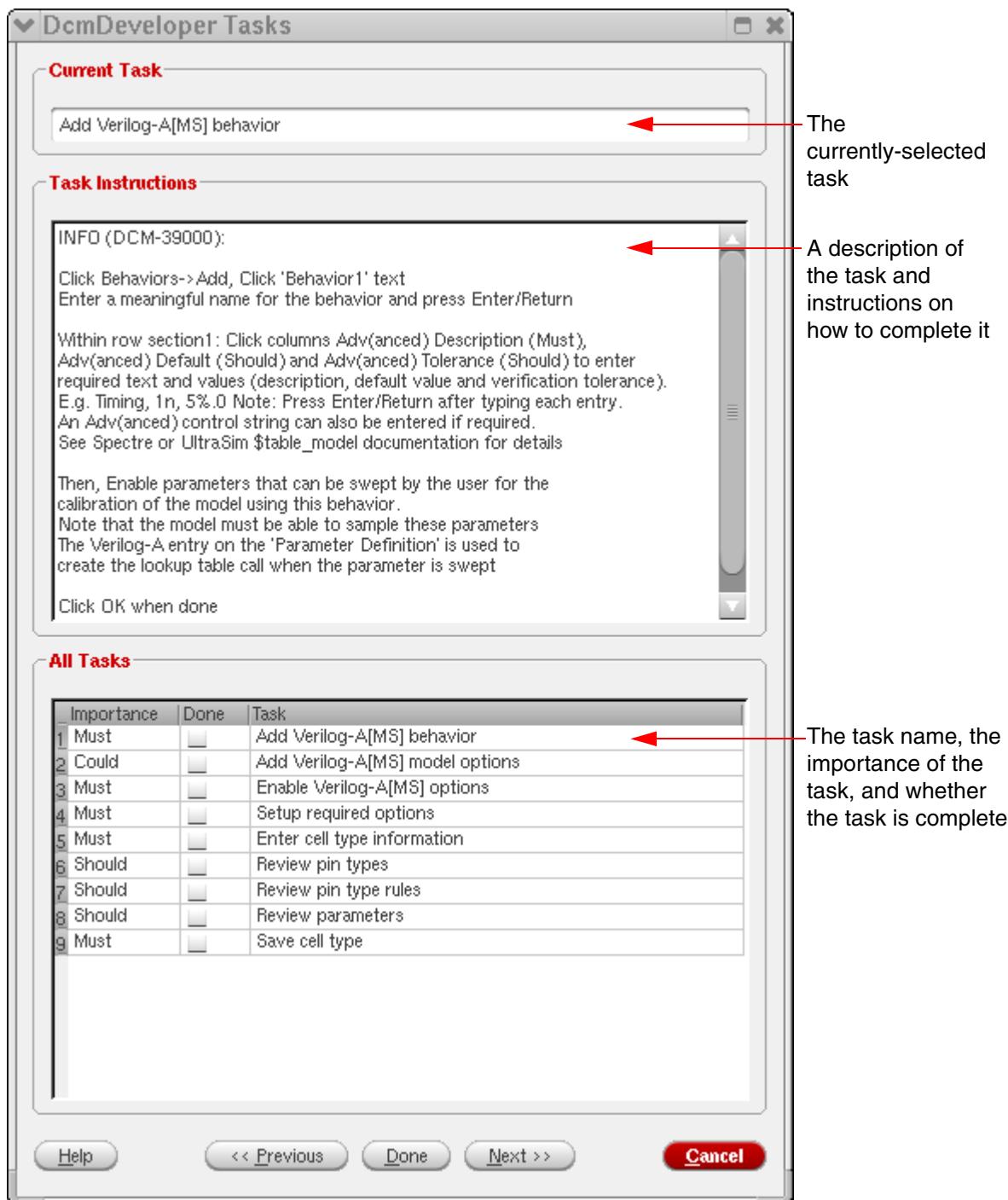
To view the Tasks form at any time, do one of the following:

- Click *OK* when finished with the Wizard.
- Choose *File—Show Tasks* in dcmDeveloper.

# Virtuoso Analog Design Environment GXL User Guide

## Cell Type Development

The Tasks form appears.



## **Marking a Task Complete**

To mark a task complete, do one of the following:

- Select the *Done* check box in the All Tasks table.
- Click *Done*.

The Task form displays the next unfinished task.

## **Viewing the Next Unfinished Task**

To view the next incomplete task:

- Click *Next*.



***The list selection behavior does not wrap around, so the selection of the next task will stop at the last task on the list.***

## **Viewing the Previous Unfinished Task**

To view the previous incomplete task:

- Click *Previous*.



***The list selection behavior does not wrap around, so the selection of the previous task will stop at the first task on the list.***

## **Closing the Tasks Form**

To close the task form:

- Click *Cancel*.

## Creating a New Cell Type

If you have been working on one cell type, and now you want to create another one, do one of the following:

- Select the *Create cell type* check box on the Cell Type form, then click *OK*.
- Choose *File — New*.

All information is cleared from the dcmDeveloper window so that you can begin creating a new cell type.

You can now:

- Add and Configure test benches (not required for top-down models)
- Add Models (not required for design verification)
- Specify Cell Type Generation Options
- Specify the Output Destination and Options

## Adding and Configuring Test Benches

You have two options for adding and configuring test benches. You can manually specify the test instances and states, or you can load an ADE XL view, from which dcmDeveloper determines the test instances, states, and DCM parameters. You can then further modify the data as outlined in the following sections.

### Loading an ADE XL View

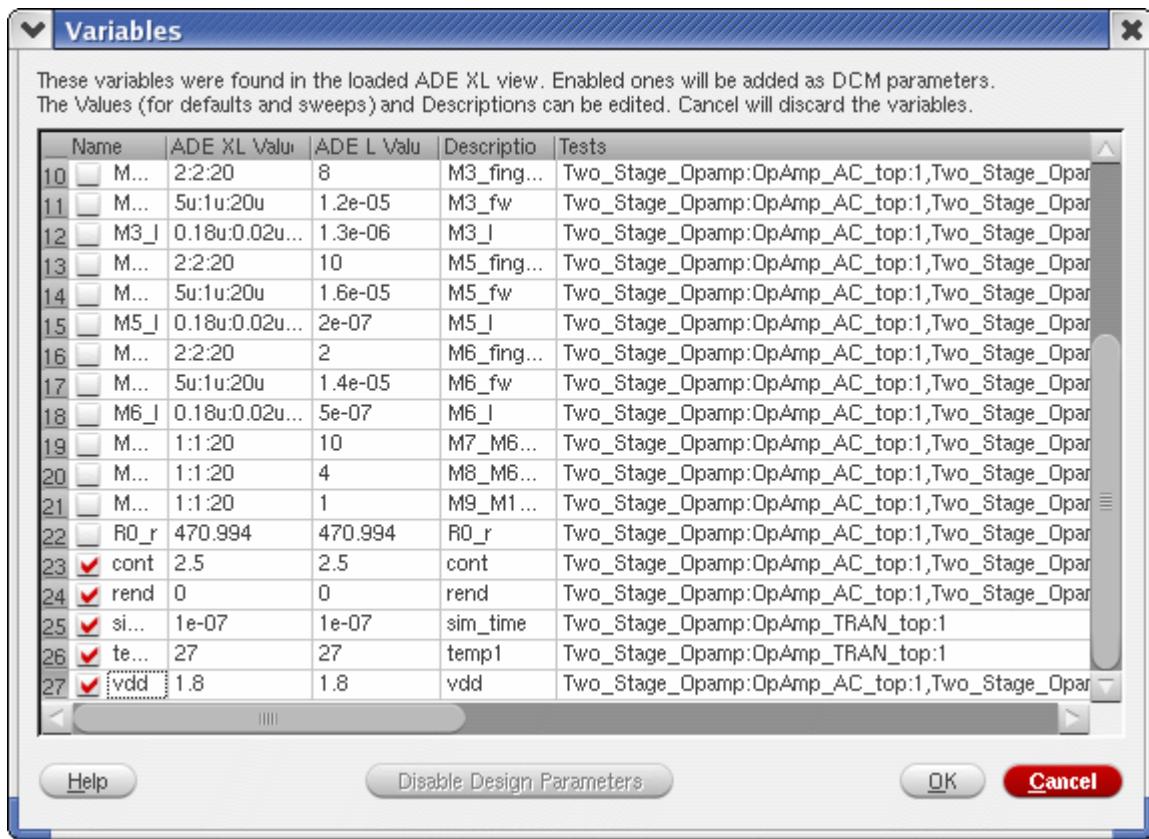
When you load an ADE XL view, dcmDeveloper:

- opens the ADE XL view and retrieves the list of tests
- adds each test to dcmDeveloper
- adds the schematic and state from ADE XL to the test
- processes the schematic into instances, then processes the state (extracting the outputs and adding them as measure instances), and
- extracts the ADE XL variables (optionally pruned to remove the design specific ones, this pruning is done automatically if the design has already been specified)

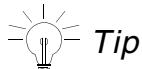
To load an ADE XL view:

1. Choose File — Open ADE XL.  
The Select ADE XL View to Load form appears.
2. Choose a library, cell, and view, then click *OK*. The view type must be *adexl*.  
dcmDeveloper displays warnings found during the process, then the calculator expressions from one of the test benches attached to the specified state.
3. All calculator expressions are selected by default. Disable the check boxes for any expressions you do not want added.
4. Modify the expression names, expressions, and descriptions as desired.
5. Click *OK* in the Calculator Expressions form.
6. The calculator expressions for any additional test benches are displayed. Repeat step 3 through step 5 for all test benches.

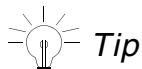
The Variables form appears, listing all of the variables from the specified view. The variables used within testbenches are automatically selected, while all design variables are automatically disabled.



7. Enable the check boxes for variables you want added as DCM parameters, and disable the check boxes for variables you do not want added.



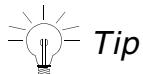
*Tip*  
The default enabled/disabled variables are recommended. A variable that is specific to the design should not normally be enabled as the cell type must work with different designs.



*Tip*  
Options for enabling and disabling all variables are available by right-clicking in the table.

8. Modify the variable values and descriptions as desired.

9. Click **OK** in the Variables form.



*Clicking *Cancel* will discard all variables.*

The test information is displayed in the Tests tab.

## **Adding the Test Benches**

1. Ensure the Tests tab is selected.
2. In the Test field, enter a name for the test.
3. Click *Add*.
4. Ensure the Instances tab is selected.
5. Choose a test bench type from the *Lib/Cell/View* drop-down list.
6. Click ... to choose the test bench.

**Note:** If a test bench has already been loaded, you are prompted to specify whether you want to append or replace the test bench instance.

# Virtuoso Analog Design Environment GXL User Guide

## Cell Type Development

The test bench data is loaded and displayed in the list box.



7. If desired, modify the existing test bench instances, pin types, and parameters by clicking *Show Advanced*.

For more information, see:

- [Adding a Measure](#)
- [Adding Instance Text](#)
- [Adding Pin Types to be Iterated](#)
- [Adding a Condition](#)
- [Moving Instances or Models into a Condition](#)
- [Modifying an Instance](#)
- [Adding and Modifying Pin Types](#)
- [Automatically Replacing Pin Types](#)
- [Adding and Modifying Parameters](#)
- [Viewing the Test Instance](#)

8. Repeat steps [step 2](#) through [step 7](#) for all tests.

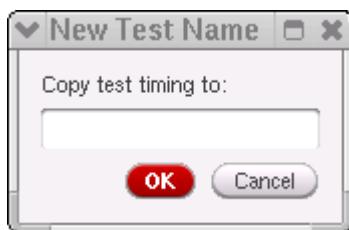
You can now load state files. For more information, see [Loading State Files](#).

## Copying a Test

You can copy a test to a different name, then modify it:

1. Select the name of the test to be copied.
2. Choose *Edit — Copy Test*.

The New Test Name form appears.



3. Enter a name for the new test in the field.
4. Click *OK*.

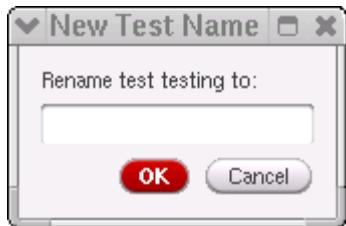
A new test with the specified name is created and displayed in the Tests Instances page; all tests are available from the dropdown menu at the top right of the page. You can modify this test using the procedures outlined in [Adding and Configuring Test Benches](#).

## Renaming a Test

You can rename a test:

1. Select the name of the test to be renamed.
2. Choose *Edit — Rename Test*.

The New Test Name form appears.



3. Enter the new name for the test in the field.
4. Click *OK*.

The new test name is displayed in the dropdown menu at the top right of the Tests Instances page. You can modify this test using the procedures outlined in [Adding and Configuring Test Benches](#).

## Deleting Tests

If you no longer want a test, you can delete it. You have two options for deleting tests: you can [delete a single test](#), or you can [delete all tests](#).

### Deleting a Test

To delete a test:

1. Choose the test from the dropdown menu at the top right of the Instances page.
2. Click *Delete*.

A dialog appears, prompting you whether you want to delete the test.

3. Click Yes to delete the test.

### **Deleting All Tests**

To delete all test:

- Choose *Edit — Delete All Tests*.

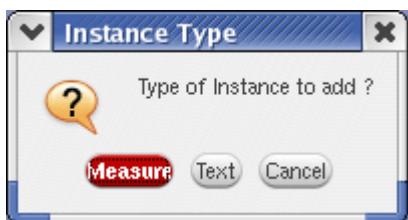
## Adding a Measure

You can add measures using the Measure Instance form. These are then written out using a COMPONENT Block for each instance of the measure.

To add a measure:

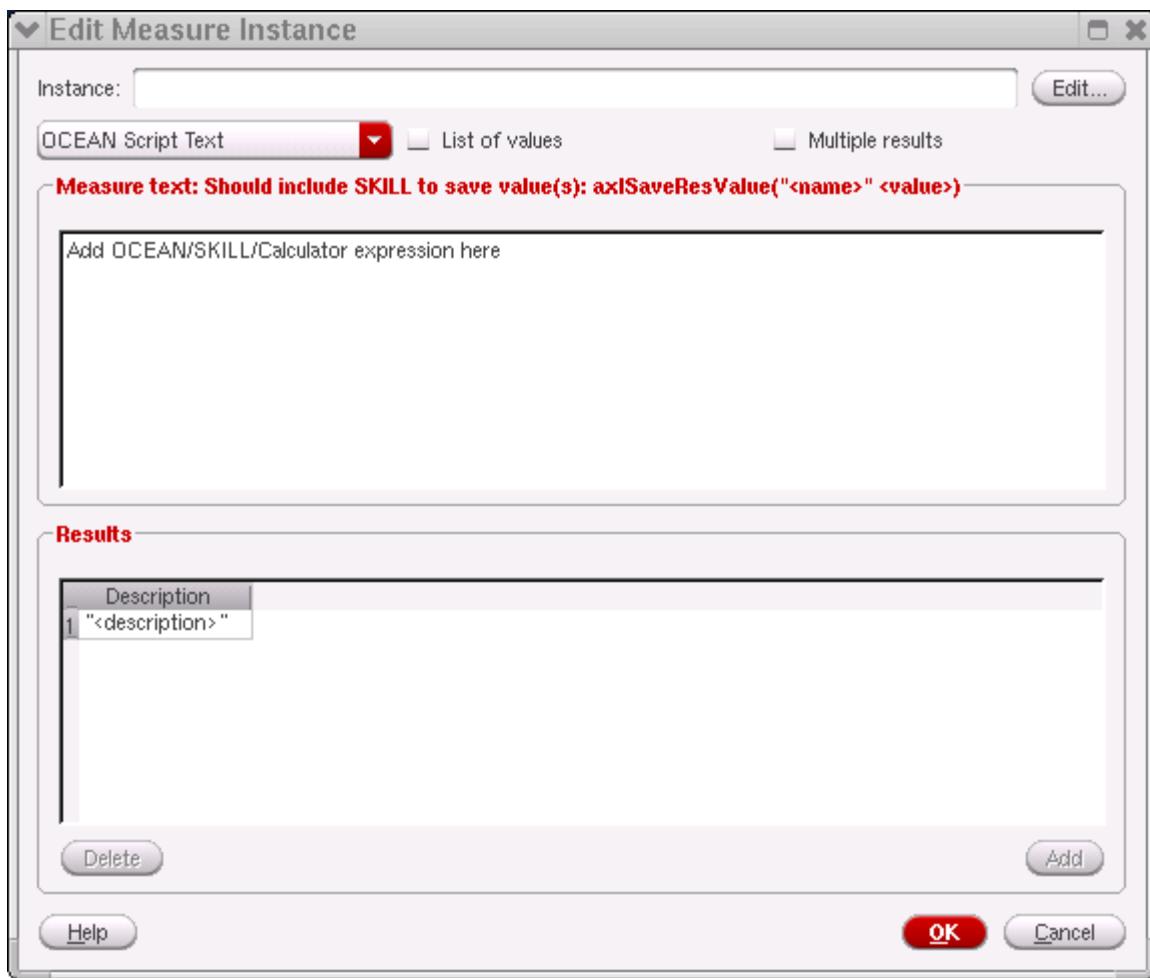
1. Choose *Edit — Add*.

The Instance Type form appears.



2. Click *Measure*.

The Edit Measure Instance form appears.



3. Enter the instance name in the Instance field.



**If the measure instance is iterated with a DCM\_GENERATE, the instance name should contain references to the pins in the DCM\_GENERATE statement. For more information, see step 1 through step 5 in Modifying an Instance.**

4. Specify the measure.

For more information, see:

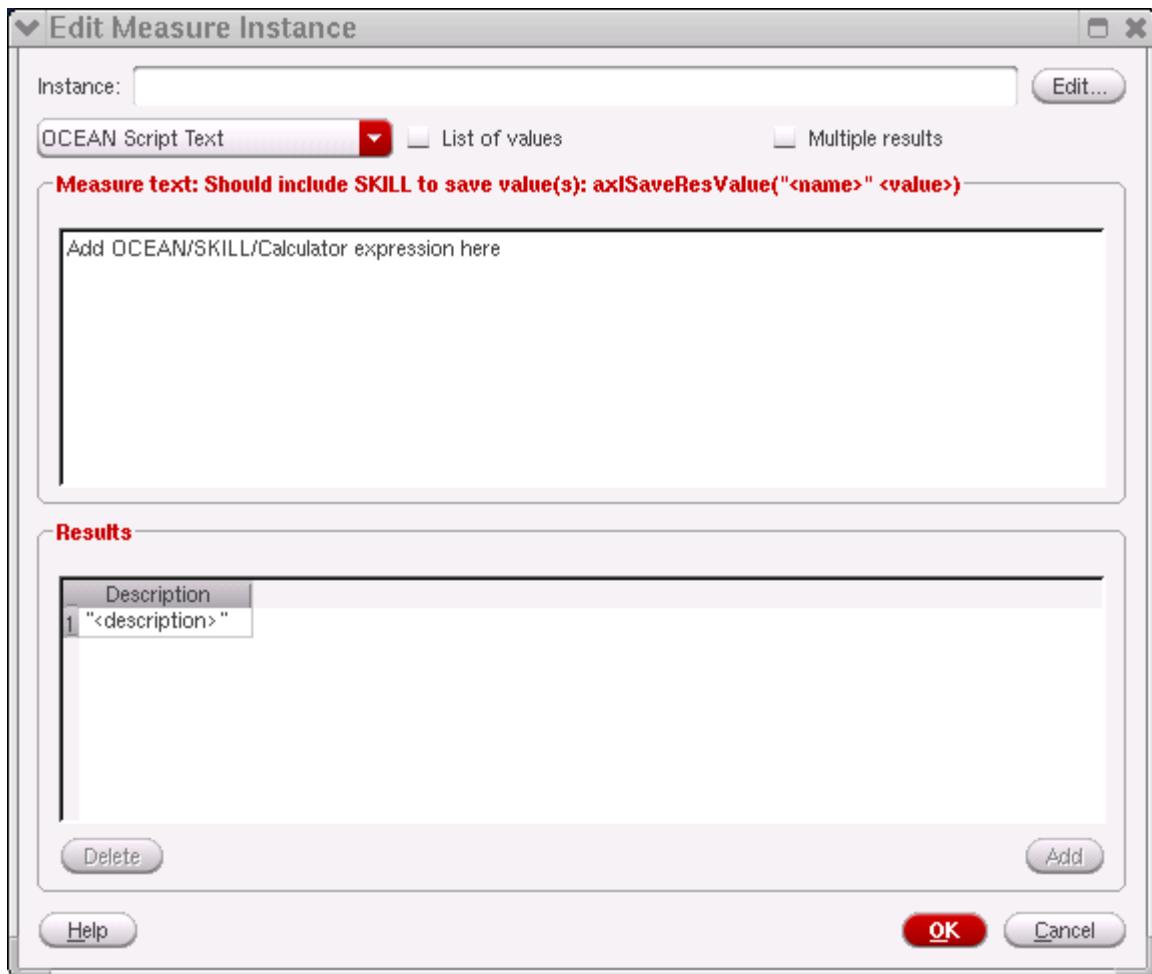
- [Specifying an Ocean Script Text](#)
- [Specifying a Calculator Expression](#)

- Specifying a Simulator Measure.
5. Click *OK* to save your measure.

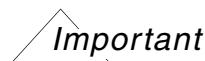
## Specifying an Ocean Script Text

To specify an Ocean script for a measure:

1. Choose *Ocean Script Text* from the drop-down list at the top.



2. Enter the Ocean script text in the Measure text field.



Ensure that the Ocean script text includes SKILL code to save the value(s):  
axISaveResValue ("<name>" <value>).

3. If the measure result will be a vector rather than a scalar value, select the *List of Values* check box.

4. If the script returns multiple results, select the *Multiple Results* check box.

If this check box is selected, the measure returns multiple results with the name <instance>:<result name> in the result database. If this check box is not selected, there will be a single measure named the same as the instance name.

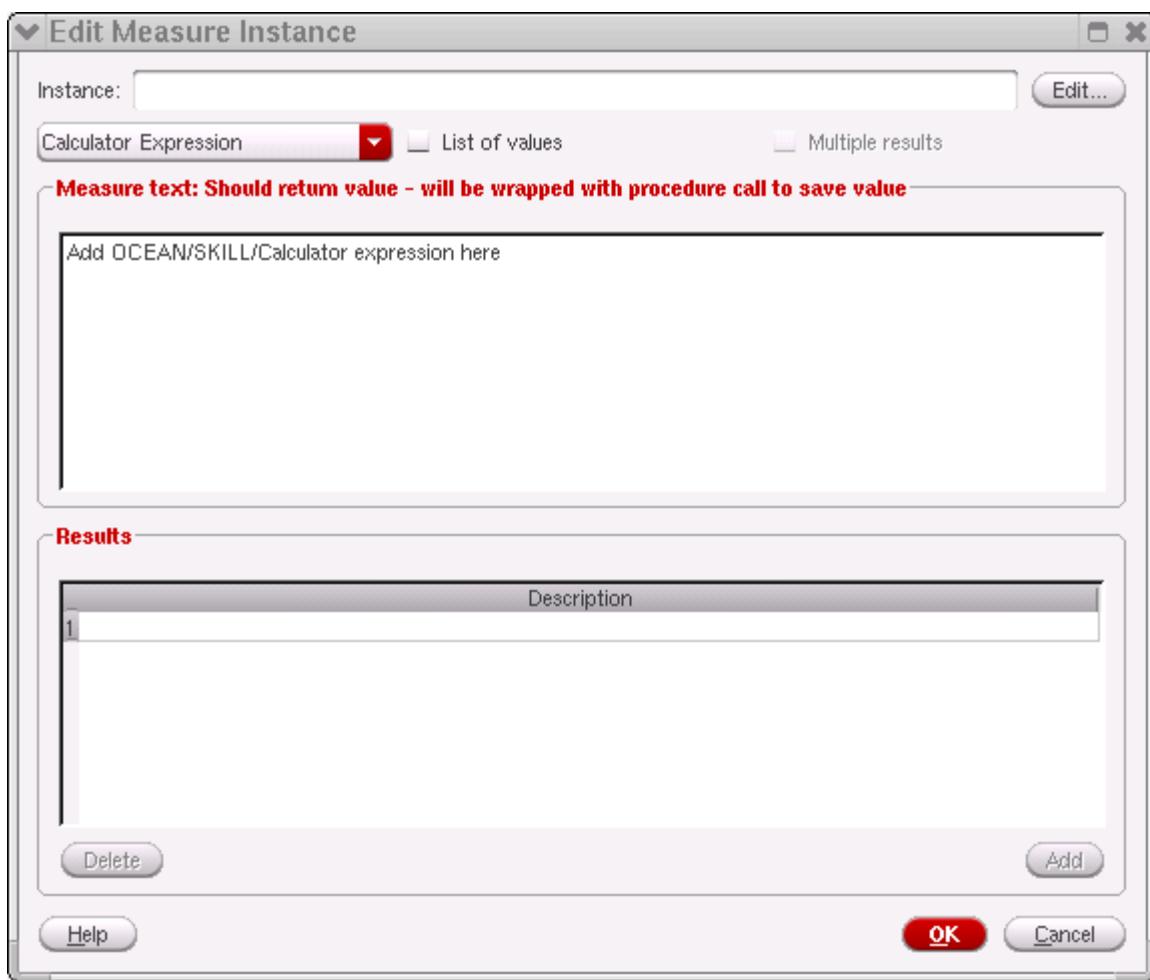
5. Enter result descriptions in the Results table.

- ❑ If the Ocean script returns a single result, enter a description for the result in the Results table.
- ❑ If the Ocean script returns multiple results, enter a name and description for the first result in the Results table. Click *Add* to create more result descriptions. To delete any result descriptions you no longer want, select the result, then click *Delete*.

## Specifying a Calculator Expression

To specify a calculator expression:

1. Select *Calculator Expression* from the drop-down list.



2. Enter the calculator expression in the Measure Text field.

 **Important**

The calculator expression will be wrapped in a procedure call to save the returned value.

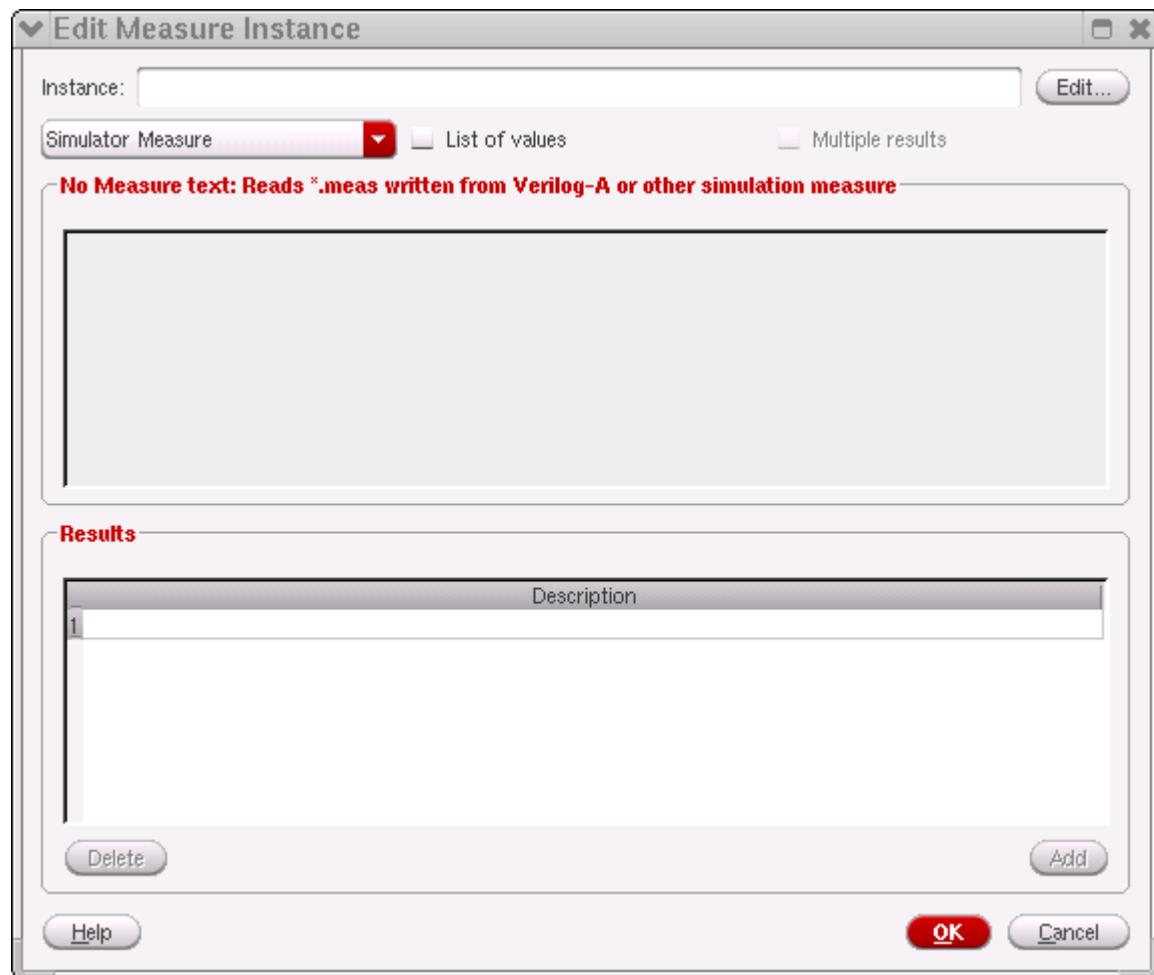
3. If the measure result will be a vector rather than a scalar value, select the *List of Values* check box.
4. Enter a description for the returned result in the Results table.

## Specifying a Simulator Measure

You can also utilize measurements run during simulation. DCMDeveloper looks for the instance of Verilog-A or VHDL-AMS on the testbench that matches the specified measurement instance name.

To read a measure output from Verilog-A or another simulator measure:

1. Select *Simulator Measure* from the drop-down list.



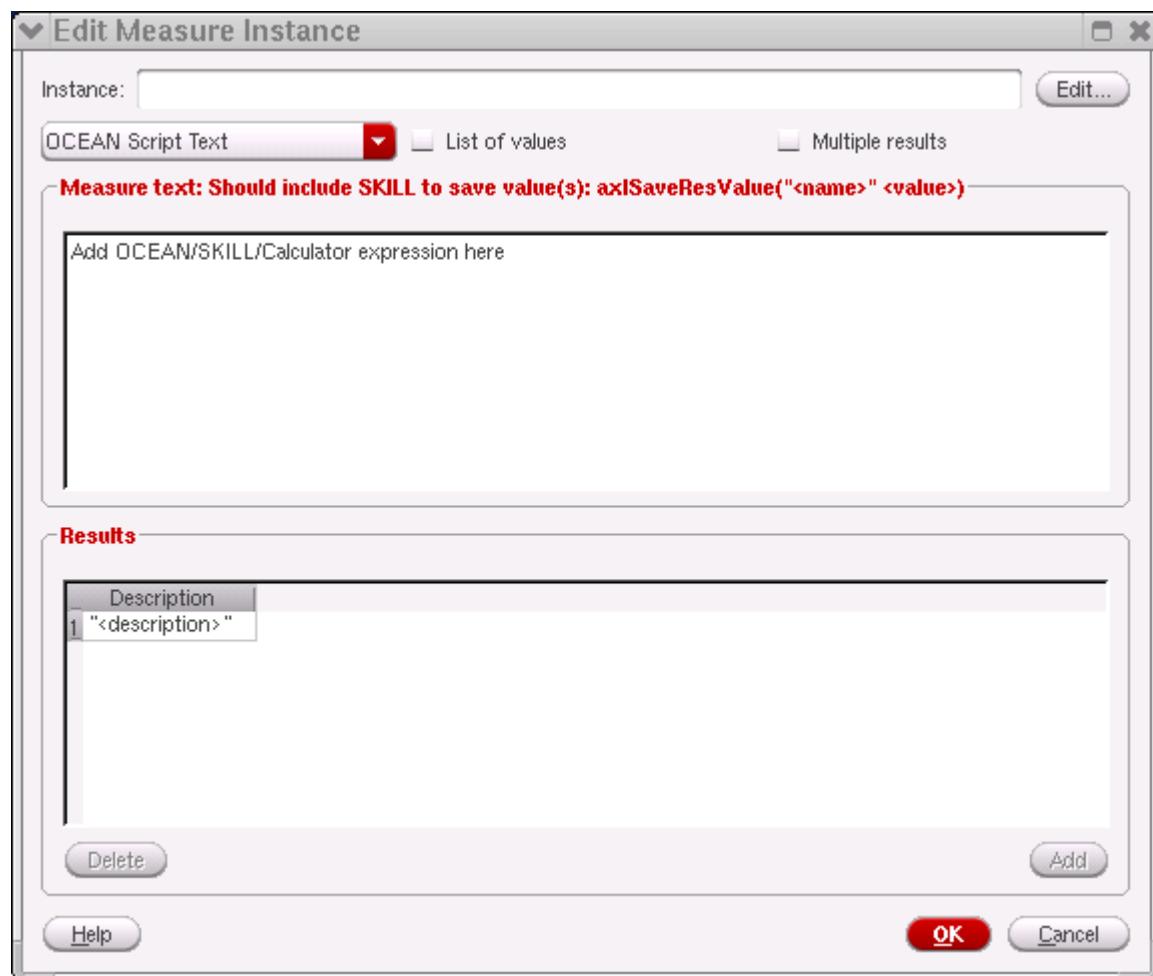
2. If the measure result will be a vector rather than a scalar value, select the *List of Values* check box.
3. Enter a description for the returned result in the Results table.

## Modifying a Measure

To modify an existing measure:

1. Double-click the measure instance in the test bench list box, or select it, then choose *Edit*—*Edit*.

The Edit Measure Instance form appears.



2. Modify the values as described in [Adding a Measure](#).
3. Click *OK* when finished.

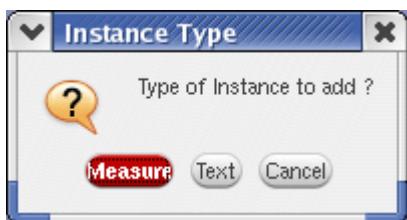
## Adding Instance Text

Instance names must be unique. As a result, if an instance is iterated with DCM\_GENERATE, the instance name should include the pin types being iterated. In addition, instance text is useful for the OpenDCM macros for which assistants are not yet available, and for entering Spectre instances. To enable these instances for use in assistants, save the cell type, then load it again.

To add instance text:

1. Choose *Edit — Add*.

The Instance Type form appears.



2. Click *Measure*.

The Edit Instance text form appears.



3. Enter the instance text in the field.

4. Click *OK* to save the instance text and display it in the main form.

## **Adding Pin Types to be Iterated**

You can add a DCM\_GENERATE function if you want to iterate a set of pins of a particular type in a test or model template and then generate instances for any specified components and measures.

1. In the list box on the Tests tab, select either one or more contiguous instances in a test, or one or more contiguous lines of text in a model.
2. Choose *Edit — Add Generate*.

The Generation form appears.



3. In the Pin Types list box, select the check box for the pin(s) to be iterated.

The DCM\_GENERATE condition is applied.

4. Click *OK*.

Once you have created the DCM\_GENERATE condition, you can move instances or models so that they utilize it. For more information, see Moving Instances or Models into a Condition,

## Adding a Condition

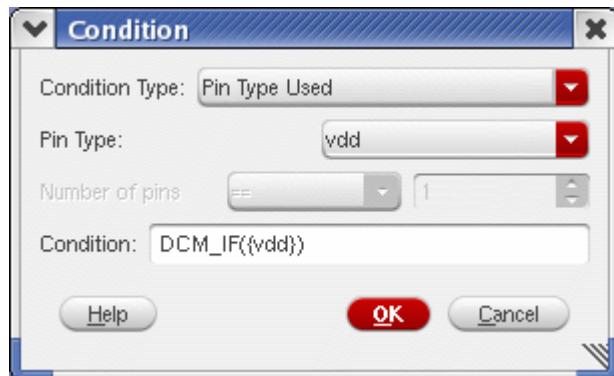
You can add a condition for pin types used. You can also add a condition specific to the different Verilog and Liberty languages.

These conditions are written to the templates using the [DCM\\_IF Block](#).

To add a condition:

1. In the list box on the Tests tab, select either one or more contiguous instances in a test, or one or more contiguous lines of text in a model.
2. Choose *Edit — Add Condition*.

The Condition form appears.



## Adding a Pin Type Usage Condition

To add a pin type usage condition:

1. In the Condition form, choose one of the following from the *Condition Type* drop-down list:
  - a. Pin Type Used
  - b. Pin Type Not Used
  - c. Inverted Pin Type Used
  - d. Non Inverted Pin Type Used
  - e. Number of Pins of Type Used
2. Choose the pin type from the *Pin Type* drop-down list.

3. If necessary, choose the number of pins from the *Number of pins* drop-down list.

The resulting condition is generated in the Condition field.

### **Adding a Language Condition**

To add a condition specific to a language:

- In the Condition form, choose one of the following from the *Condition Type* drop-down list:
  - a. Verilog-A[MS] Dynamic Vdd
  - b. Verilog-A[MS] Model Output Voltage
  - c. Liberty Power
  - d. Verilog-D Timing Constraints

The resulting condition is generated in the Condition field.

## **Moving Instances or Models into a Condition**

If you have created a DCM\_GENERATE or DCM\_IF condition, and you want to add instances or models to them, you can utilize the Move Up and Move Down features.

To move an instance or model within a DCM\_GENERATE or DCM\_IF condition:

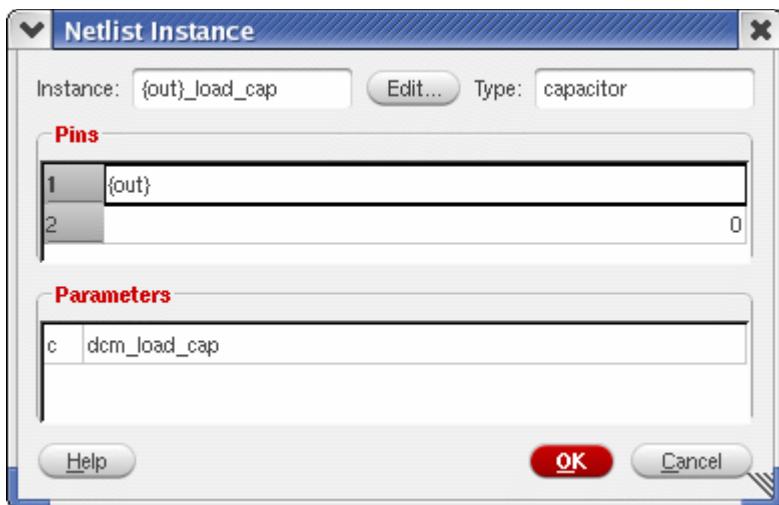
1. Select the instance or model you want to move.
2. In the advanced options, click one of the following buttons until the instance or model is where you want it:
  - Move Up
  - Move Down

## Modifying an Instance

To modify the pin type, instance name, or parameters for an instance:

1. Right-click on the instance name and choose *Edit*.

The Edit Instance form appears.



2. Click *Edit* to modify the instance name or type.

The Edit Instance Name form appears.



3. Choose a pin type from the *Pin Type* drop-down list.
4. Enter an instance name in the *Instance* field, or if you want to replace part of the name with a pin type, highlight the portion of the text and click *Replace*.
5. Click *OK* to dismiss the Edit Instance Name form.
6. In the Netlist Instance form, click on a parameter name in the Parameters list box to modify parameters.

For more information on parameters, see [Modifying Parameters](#).

## **Virtuoso Analog Design Environment GXL User Guide**

### Cell Type Development

---

7. Click *OK* when finished modifying the instance.

## Adding and Modifying Pin Types

You can add, delete, and modify pin types. A pin type is an abstract idea that allows design-neutral tests and models to be created. You can create a new pin type, then replace design-specific pin names with the neutral pin type names. For more information, see [Automatically Replacing Pin Types](#).

Pin type definitions are written to the `pinTypes` section of the `function` block of the `.gui` file: For more information, see [Pin Type Definitions](#).

For more information, see:

- [Adding a New Pin Type](#)
- [Modifying Pin Types](#)
- [Deleting a Pin Type](#)

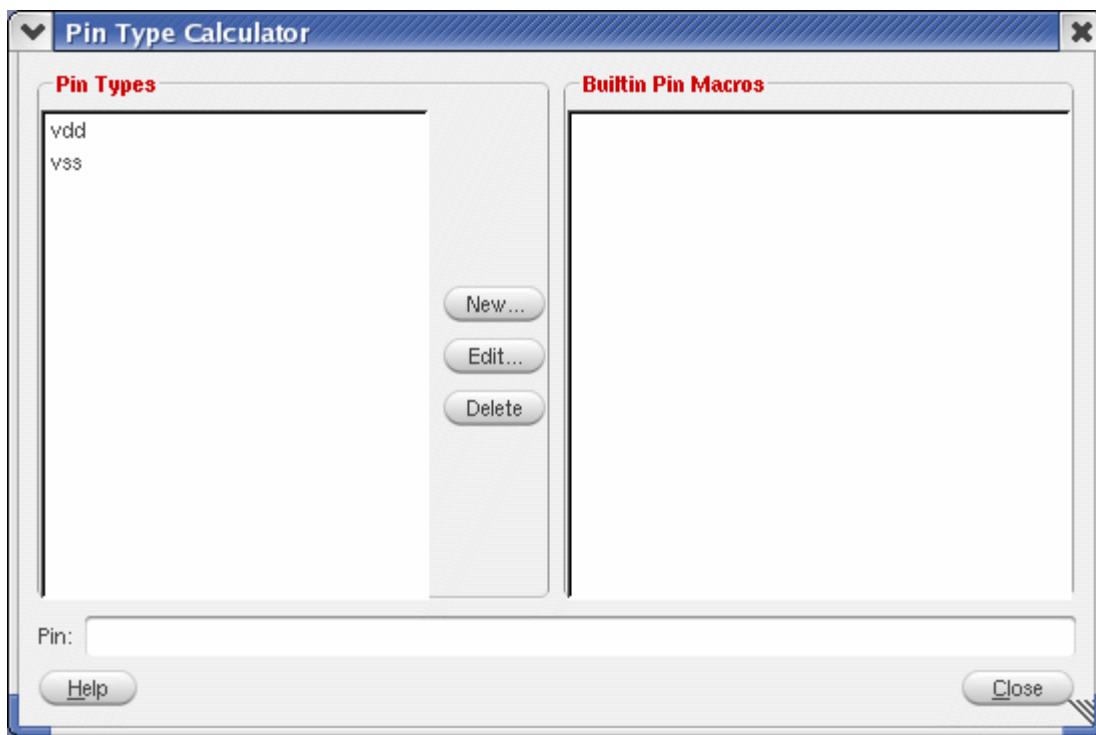
### Adding a New Pin Type

You can add a new pin type. Note that if you are defining both inverted and non-inverted pin types, they should both have the same name. Marking the pin as inverted adds an `:I` suffix to the pin name. As a result, the same pin type can be used within the template, with the `DCM_INV` macro used to determine if the design pin was set to an inverted or a non-inverted pin type.

To add a new pin type:

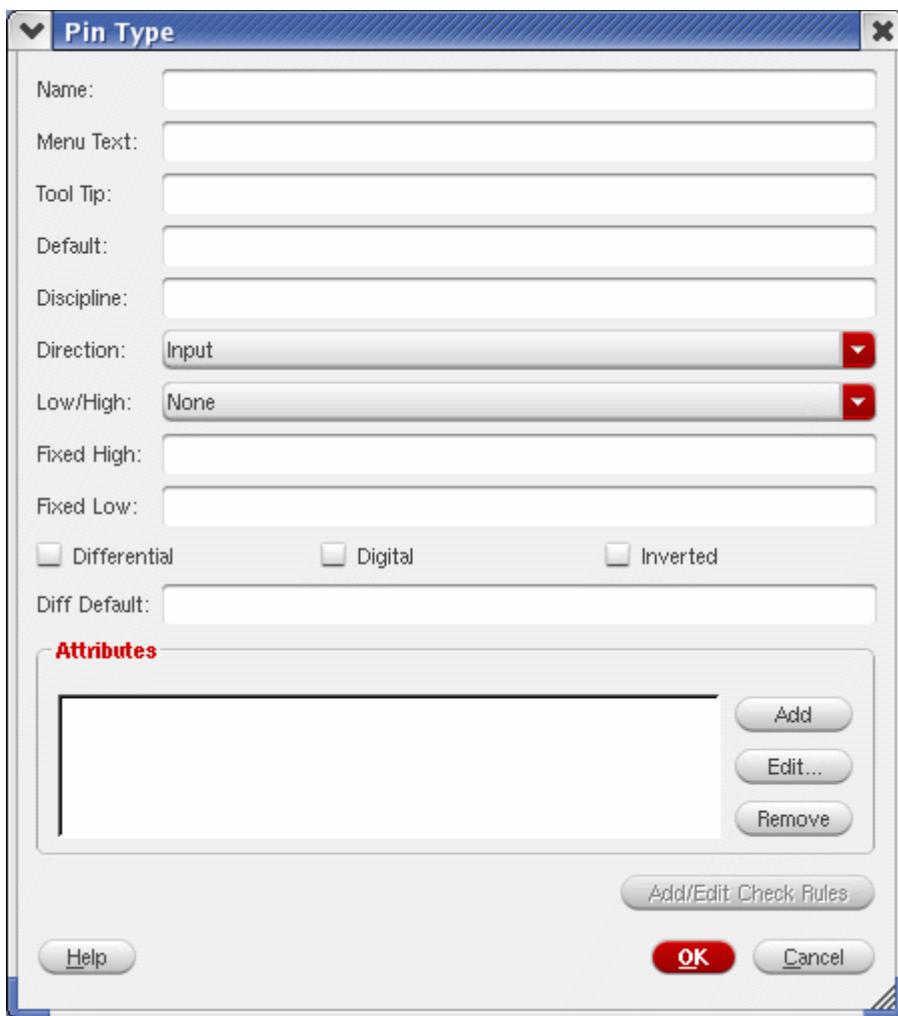
1. Choose *Tools — Edit Pin Types*.

The Pin Type Calculator appears.



**2.** Click *New*.

The Pin Type form appears.



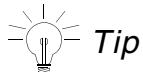
3. Enter a pin name in the Name field.
4. Enter text for this menu item in the Menu Text field.
5. Enter a tool tip in the ToolTip field.
6. Enter a default pin name in the Default field.

This default pin type defines which design pin names are automatically defaulted to this pin type. For example, if all pins beginning with "cp.\*" should be defaulted to a Clock pin type, then the Default field should contain "cp."

7. (Verilog-A[MS] only) Enter a discipline in the Discipline field.

This field defines the discipline type for pins when the model is created. The default discipline is negative. You can specify "none" for pins not utilizing a discipline.

8. Choose a pin direction from the *Direction* drop-down list.
9. Choose a Low/High value from the *Low/High* drop-down list.
10. Enter a fixed high value in the Fixed High field.
11. Enter a fixed low value in the Fixed Low field.
12. Select the *Differential* check box if this pin type supports differential pins.
13. Select the *Digital* check box if this pin type supports digital pins.
14. Select the *Inverted* check box if this pin type supports inverted pins.



*Tip*

Marking the pin as inverted adds a :l suffix to the pin name.

15. Enter a default value for the differential pin type in the Diff Default field.
16. Click *Add* under *Attributes* to add pin attributes.

The Pin Attribute form appears. The Pin Attribute form is used to define information the cell type user can provide about the design pin that is using this pin type., e.g. low/high voltages, thresholds, etc.



17. Enter an attribute name in the Name field.
18. Enter text for the attribute in the Text field.
19. Enter a tool tip for the attribute in the Tool Tip field.
20. Enter a minimum value for the attribute in the Minimum field.

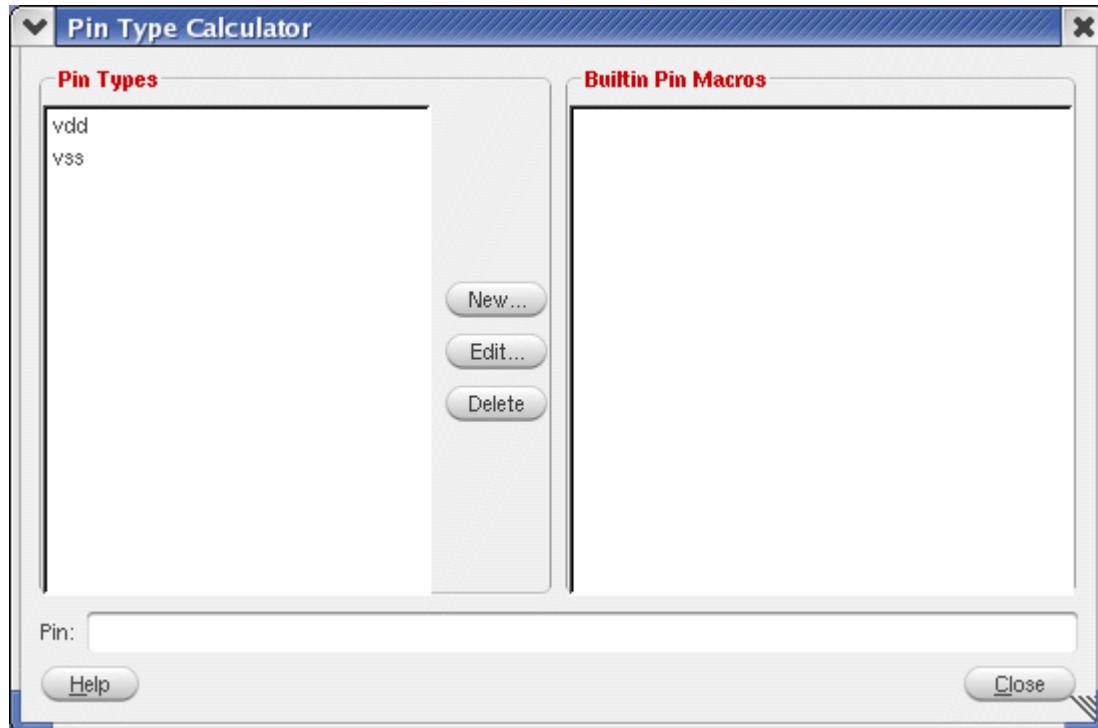
21. Enter a maximum value for the attribute in the Maximum field.
22. Choose an attribute type from the *Type* drop-down list.
23. Select the *Required* check box if this attribute is required.
24. Click *OK* to dismiss the Pin Attribute form and display the attribute name in the Attributes list box.
25. Repeat step 16 through step 24 for all attributes you want for this pin type.
26. Click *OK* to dismiss the add this pin and dismiss the Pin Type form.
27. Click *Close* to close the Pin Type Calculator form.

## Modifying Pin Types

To modify the pin types for the cell:

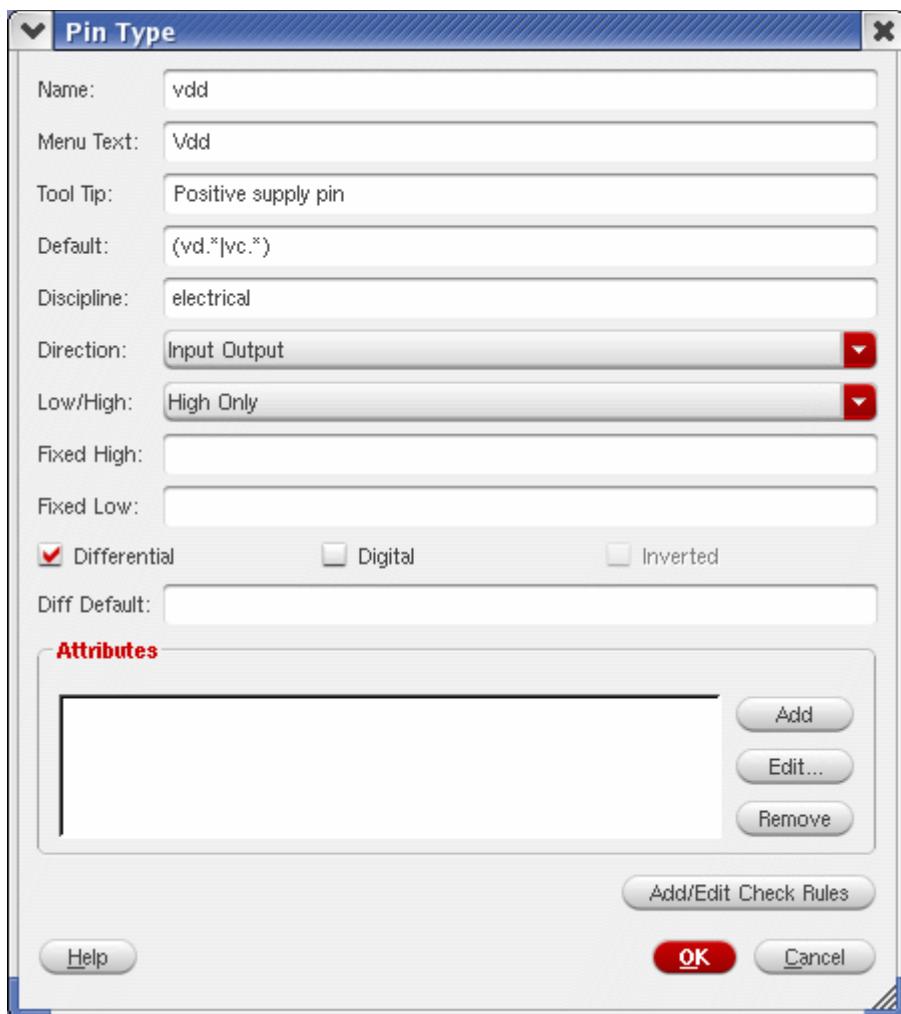
1. Choose *Tools — Edit Pin Types*.

The Pin Type Calculator appears.



2. Select the pin you want to modify.
3. Click *Edit*.

The Pin Type form appears



4. Modify any values as desired.

For more information on the parameters, see [Adding a New Pin Type](#).

5. Click **OK**.
6. Click **Close** to close the Pin Type Calculator form.

### **Deleting a Pin Type**

To delete a pin type:

1. Choose **Tools — Edit Pin Types**.

The Pin Type Calculator appears.

**2.** Select the pin type you want to delete from the Pin Types list box.

**3.** Click *Delete*.

The pin type is deleted and removed from the Pin Types list.

**4.** Click *OK* to dismiss the Pin Type Calculator form.

## Automatically Replacing Pin Types

To replace an existing pin name with a generic pin name:

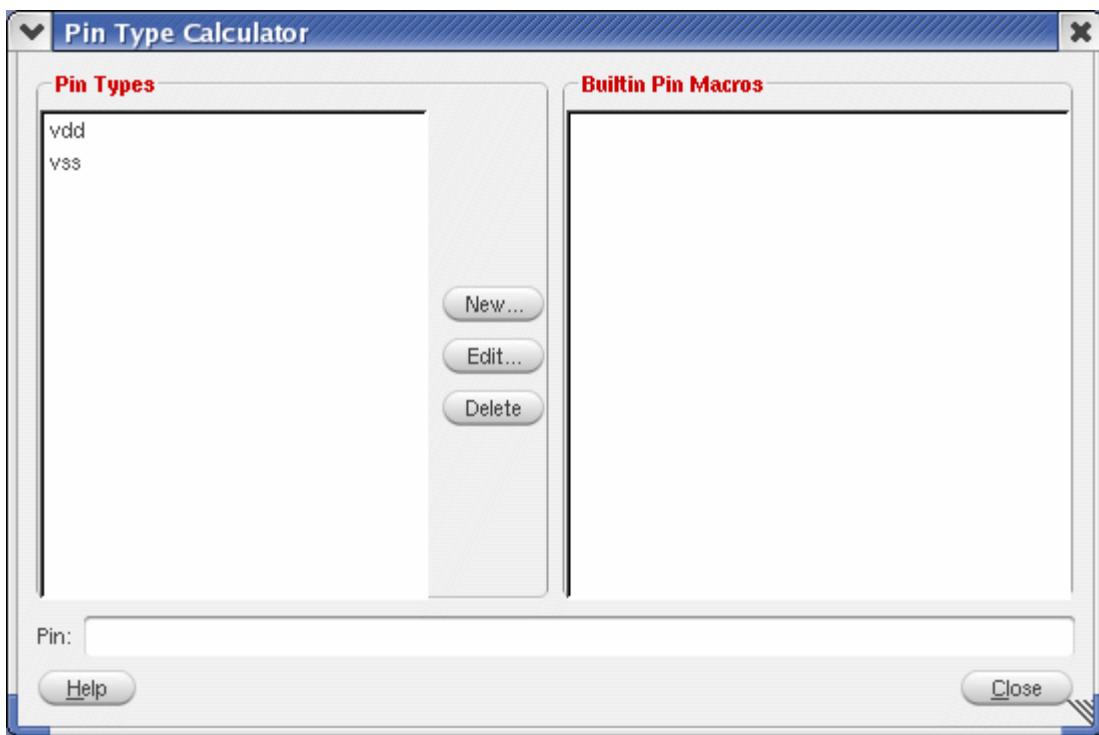
1. Choose *Tools — AutoReplace Pin Types*.

The Create Pin Types form appears



2. In the Nets list box, select the check box for the pin name you want to replace with a pin type.
3. Click *Get Pin Type*.

The Pin Type Calculator appears.



4. In the Pin Types list box, select the pin type you want to use.

To instead create a new pin type, see [Adding a New Pin Type](#). To modify an existing pin type before using it, see [Modifying Pin Types](#).

5. Click *OK* to dismiss the Pin Type Calculator form.

The selected pin type is displayed in the Pin Type field.

6. Click *OK*.

You must now modify any instance names that refer to the original design pin to instead utilize the new pin type. For more information, see [step 1 through step 5](#) in [Modifying an Instance](#).

## **Adding and Modifying Parameters**

You can add, delete, and modify parameters. Parameters, which correlate closely with ADE XL variables, are used throughout test benches to set values or sweeps.

Parameters defined here are written to the `parameters` block of the `.gui` file. For more information, see [parameters Block](#).

For more information, see:

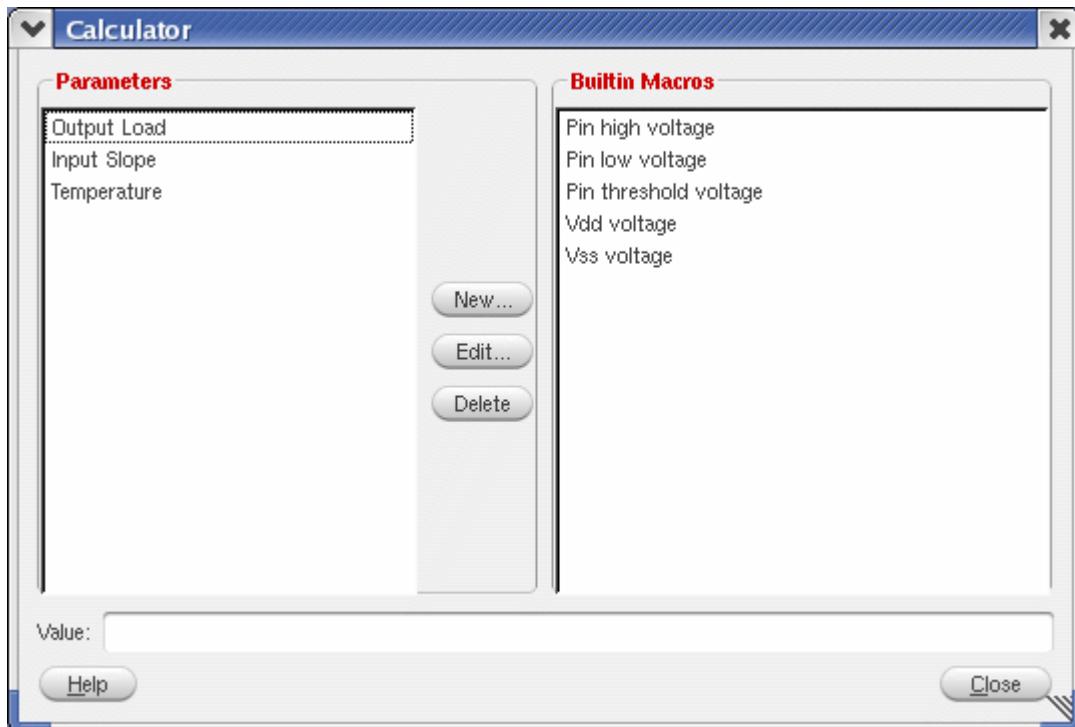
- [Adding a Parameter](#)
- [Modifying Parameters](#)
- [Deleting a Parameter](#)

### **Adding a Parameter**

To add parameters:

1. Click *Edit* under *Parameters* in the Advanced pane, or click *Edit Parameters* on the Output tab.

The Calculator form appears. Any existing parameters are listed in the Parameters list box. Built-in macros are listed in the Builtin Macros list box.



2. Click New.

The Parameter Definition form appears.



3. Enter an ID for the parameter in the Name field.

**Note:** This is the only required field in this form. However, all other entries are recommended for the ease of use when the cell type is utilized.

4. Enter the name for the ADE XL parameter in the Parameter field.

This field defaults to the Name value if left blank.

5. Enter menu text for the parameter in the Menu Text field.

This field defaults to the Name value if left blank.

6. Enter a tool tip in the Tool Tip field.

7. Enter the units value in the Units field.

8. Enter the text to be used within the Verilog-A model in the Verilog-A field.

This is the text used within the model when this parameter sweep is used to index a lookup table created using the DCM\_CALIBRATE statement.

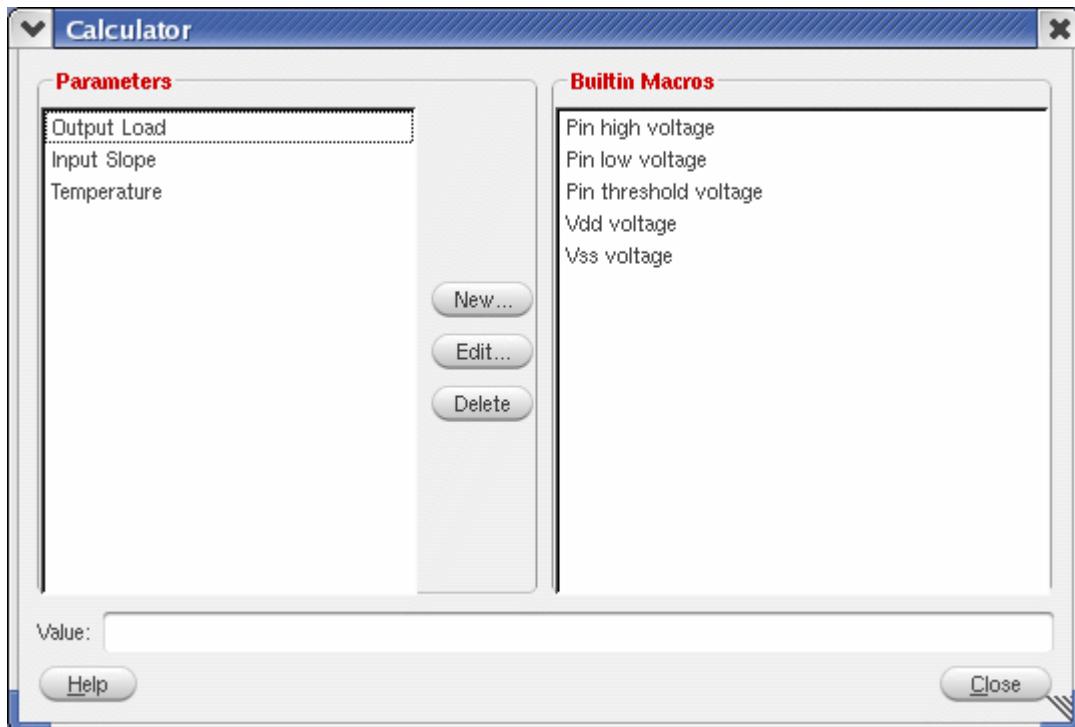
9. Choose a parameter type from the *Type* drop-down list.
10. Enter a minimum value in the Minimum field.
11. Enter a maximum value in the Maximum field.
12. If you want this parameter to include defaults for sweeps used during calibration, select the *Set Sweep Defaults* check box.
13. Specify the type of sweep default you want to set:
  - If you want to specify the defaults as a list, select the *List* check box, then enter a space or comma-separated list of default values in the field.
  - If you want to specify the defaults as a range, select the *From/To/By* check box, then specify the start, end, and step size values for the default range in the appropriate fields.
14. Click *OK* to dismiss the Parameter Definition form.
15. Click *OK* to save the parameter information and dismiss the Calculator form.

## Modifying Parameters

To edit parameters:

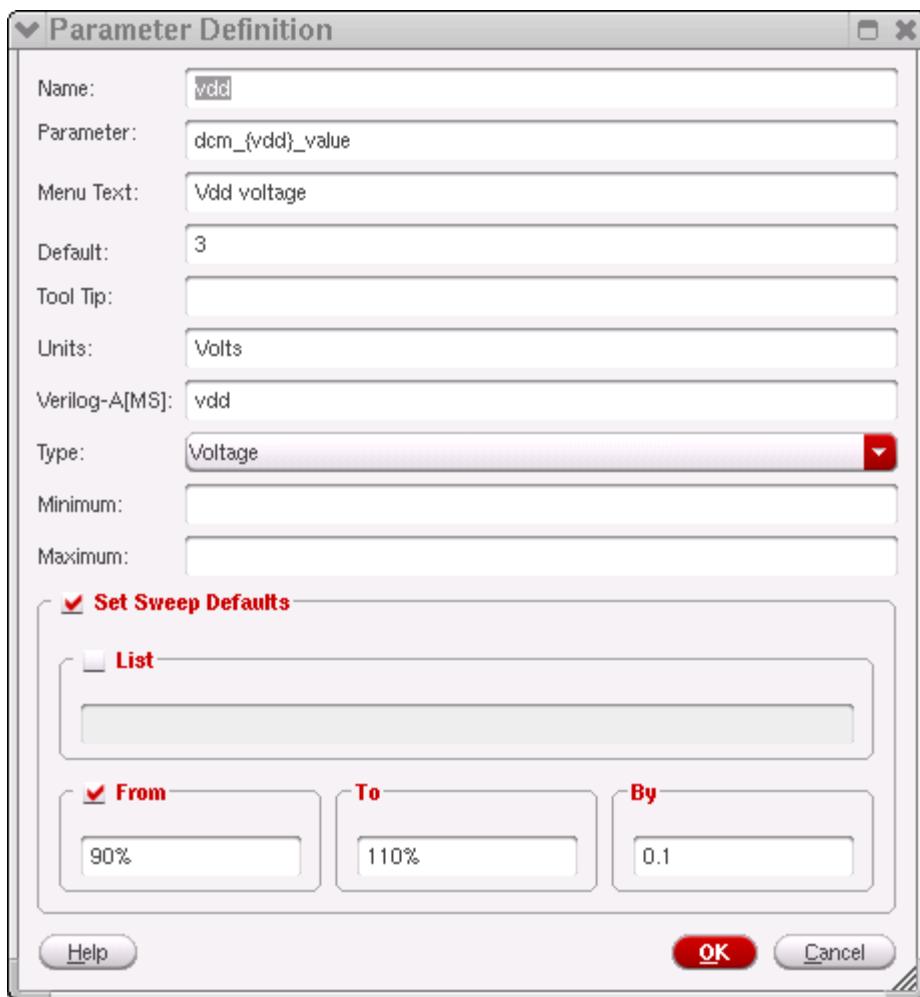
1. Click *Edit* under *Parameters* in the Advanced pane, or click *Edit Parameters* on the Output tab.

The Calculator form appears. Any existing parameters are listed in the Parameters list box. Built-in macros are listed in the Builtin Macros list box.



2. Select the parameter you want to modify from the Parameters list box.
3. Click *Edit*.

The Parameter Definition form appears.



4. Modify the fields as desired.

For more information on the form fields, see [Adding a Parameter](#).

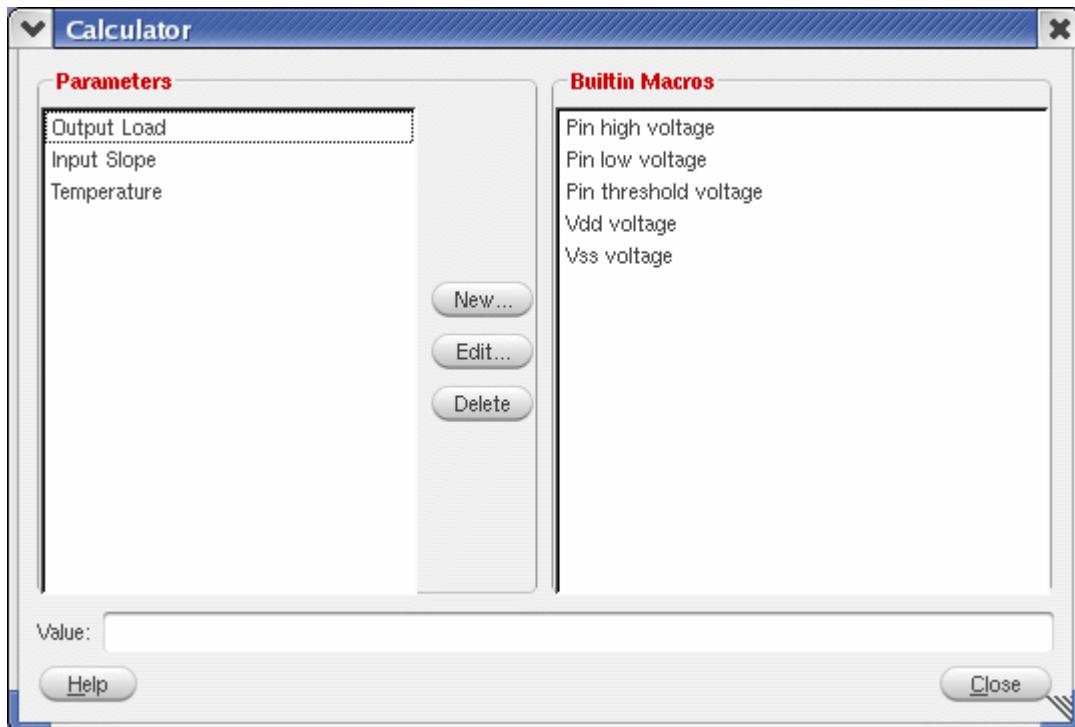
5. Click *OK* to dismiss the Parameter Definition form.
6. Click *OK* to dismiss the Calculator form.

## Deleting a Parameter

To delete a parameter:

1. Choose *Tools — Edit Parameters*.

The Calculator form appears. Any existing parameters are listed in the Parameters list box. Built-in macros are listed in the Builtin Macros list box.



2. Select the parameter you want to delete from the Parameters list box.
  3. Click *Delete*.
- The parameter is deleted and removed from the list.
4. Click *OK* to dismiss the Calculator form.

## Viewing the Test Instance

You can view the source for the instance under test. This feature is useful when there are issues parsing the file due to lost detail. Or if you have made changes, you can use this feature to view the original numbers and data.

### 1. Choose *View — Test Instance Source*.

The source for the instance under test appears.



```
DCM_FILE netlist
simulator lang=spectre

DCM_GENERATE(out) NAMEGEN_OFF BEGIN
DCM_FILE netlist
{out}.load_cap ({out} 0) capacitor c=dcm_load_cap
DCM_END_GENERATE

DCM_GENERATE({in}) BEGIN
DCM_IF(DCM_INV({in}))
DCM_DEFINE high DCM_PIN_LOW({in})
DCM_DEFINE low DCM_PIN_HIGH({in})
DCM_ELSE
DCM_DEFINE high DCM_PIN_HIGH({in})
DCM_DEFINE low DCM_PIN_LOW({in})
DCM_ENDIF
DCM_DEFINE pattern 010
{in}.source ({in} 0) vsource type=pwl wave=\[
DCM_WAVE(bits={pattern},hi={high},lo={low}),
interval=dcm_interval,rise_slope=dcm_slope,fall_slope=dcm_slope,
delimiter='\
DCM_END_GENERATE

DCM_FILE simulator Options
(opts temp) "VAR('dcm_temperature')"
DCM_FILE analyses
analysis(tran fields enable) (t)
analysis(tran fields stop) "VAR('dcm_interval')3.0"
DCM_GENERATE(out,{in}) BEGIN
DCM_FILE measures

DCM_IF(DCM_INV({out}))
DCM_DEFINE in_edge falling
DCM_ELSE
DCM_DEFINE in_edge rising
DCM_ENDIF

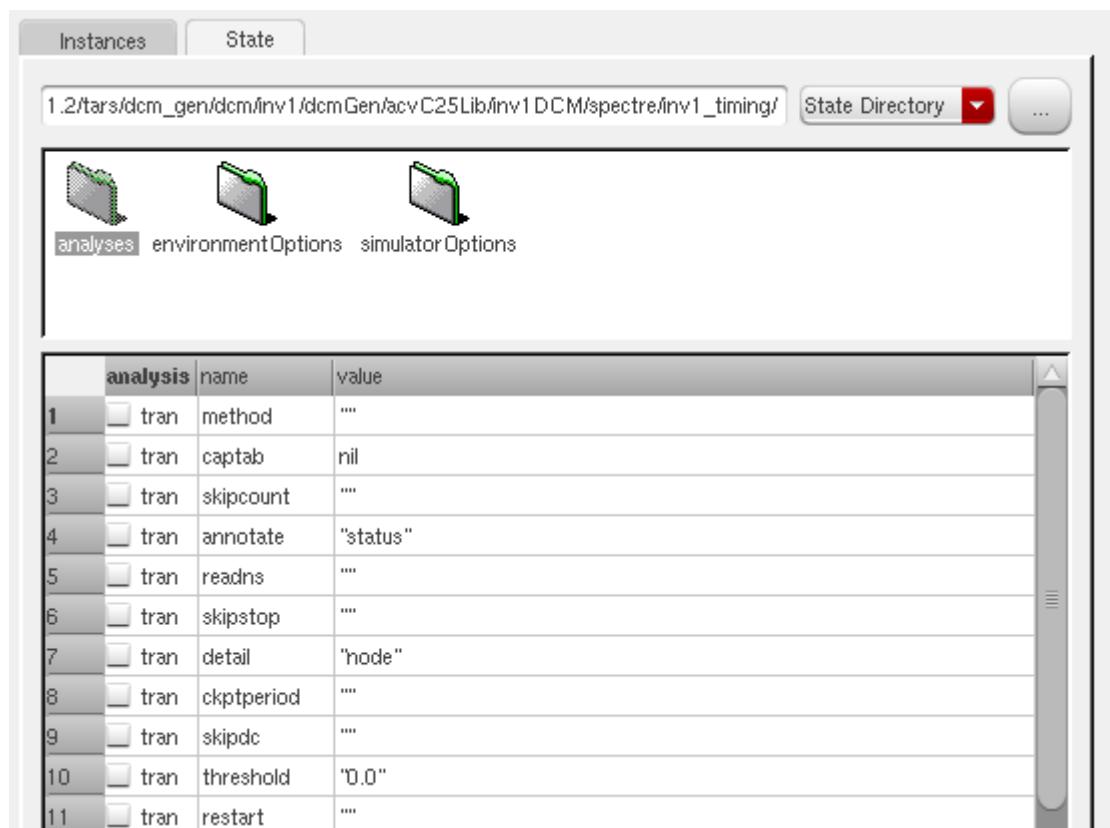
COMPONENT rise_{in}_{out} {
    _ARTIST_CALC_OCN
DCM_IF(DCM_LIB_POWER())
dcmMeasTimPwr( VT("{in}") "{in}_edge" VT("{out}") "rising" IT("{vdd[1]}.source;p") VT("{vdd[1]}") VAR("dcm_load_cap") VAR("dcm_interval") 0 0 20 80 {out}_j
DCM_ELSE
dcmMeasTim( VT("{in}") "{in}_edge" VT("{out}") "rising" VAR("dcm_interval") 0 0 20 80 'n=DCM_COMP_NAME'
DCM_ENDIF
-END_ARTIST_CALC_SECTION
RESULTS SINGLE {
    delay 'rising delay from {in,s} to {out}'
    slope 'rising slope of {out,s} (from {in,s})'
```

### 2. Click *Cancel* when finished.

## Loading State Files

1. Select the State tab.
2. Choose a state type from the *Lib/Cell/View* drop-down list.
3. Click ... to choose the state.

The state is loaded. If multiple state files are loaded, they are displayed as folders at the top.



4. If desired, modify the states.

For more information, see [Modifying the States](#).

## Modifying the States

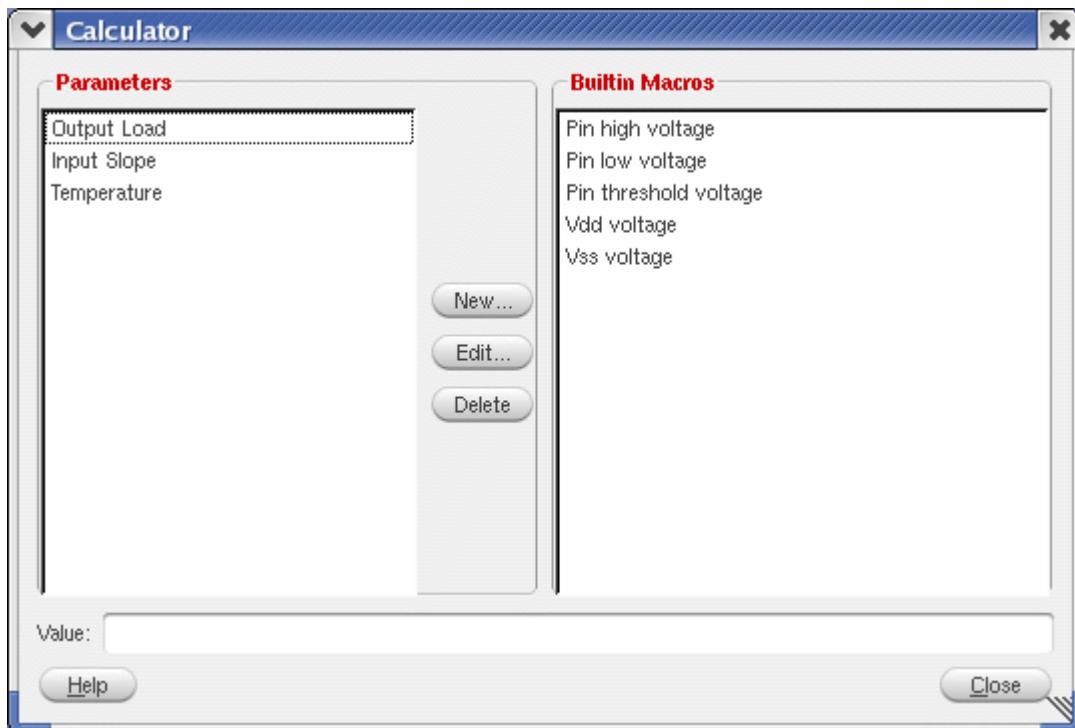
To modify a state:

1. Select a state type folder name.
2. To include an analysis, select the check box in the analysis column.

To remove an analysis, deselect the check box in the analysis column.

3. To modify a value, double-click the value column.

The Calculator form appears. The value is listed in the Value field. Any existing parameters are listed in the Parameters list box. Built-in macros are listed in the Builtin Macros list box.



4. Do one of the following:

- Select the value in the Value column and modify it.
- Select a parameter or built-in macro.

To modify or add a parameter, see [Adding and Modifying Parameters](#).

5. Click *OK* to dismiss the Calculator form.

## Adding Models

You can create and load models on the Models tab.



**Important**  
You must select the relevant Create Models option on the Options tab for the model file to be used within DCM.

To add models:

1. Select the Model tab.
2. Select the tab for the type of model to be supported with this cell type.
3. Create the model.

For more information, see [Create the Model](#).

### Create the Model

You have multiple options for creating the model. You can choose an existing model using a lib/cell/view or model file, or you can enter the model information manually.

### Loading an Existing Model

You can choose an existing model -- a lib/cell/view or model file -- by doing the following:

1. Choose a model information location option from the *Lib/Cell/View* drop-down list.
2. Click ... to browse the model file.

The model information is loaded; if model information already exists, that data is replaced.

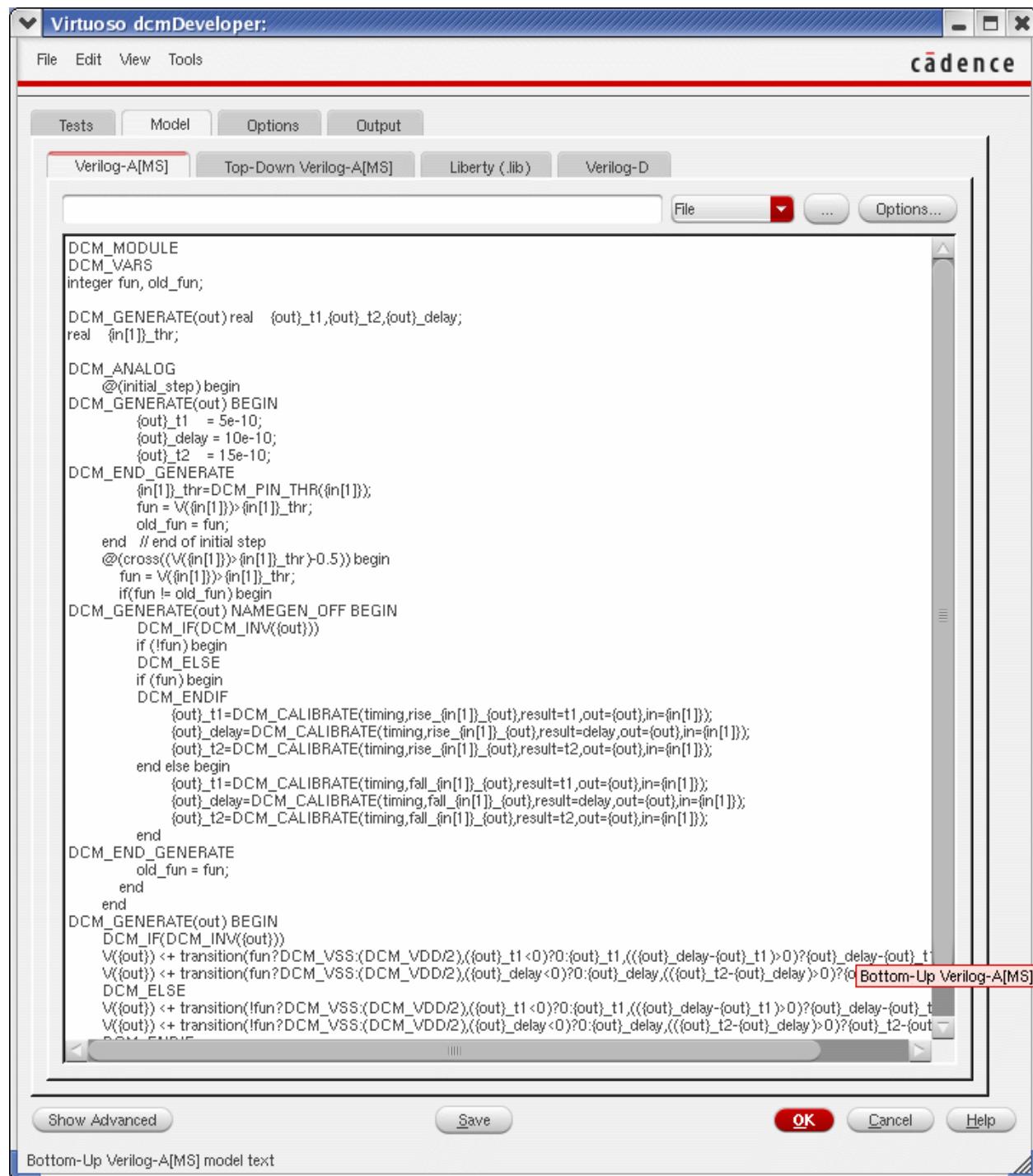


**This option differs from the RMB options Insert Lib/Cell/View and Insert File, which are used to augment model information after it is initially created. If you want to append model data, rather than replace it, utilize the RMB Insert options**

## Entering Model Information Manually

To enter the model information manually:

1. Enter the model information in the large text box, or copy and paste from a model file.



**2.** If desired, modify the model information.

- ❑ To insert additional text from a model file, right-click and choose *Insert File*. The text from the specified model file is added to the existing model information.
- ❑ To insert additional text from a lib/cell/view, right-click and choose *Insert Lib/Cell/View*. The text from the selected lib/cell/view is added to the existing model information.

**3.** If desired, modify the model options.

For more information, see [Specifying Model Options](#).

## Adding a Model Calibration Statement

You can add a **DCM\_CALIBRATE** statement. DCM\_CALIBRATE functions are used in model templates to connect the measured test results with the generated model by indicating which test and results to use when creating the calibrated table.

To add a DCM\_CALIBRATE statement to a model:

**1.** Place your cursor in the model where you would like to create the statement, or select model text you'd like to replace.\

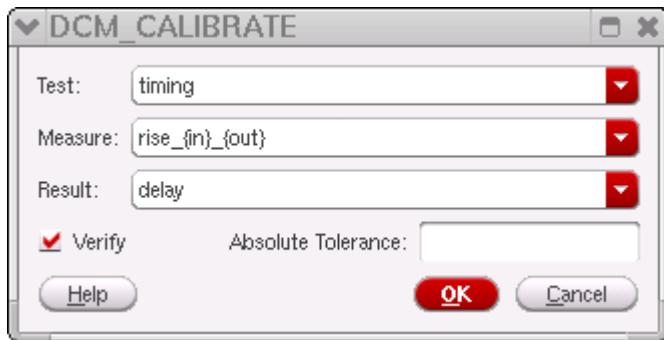


*Tip*

For example, if a variable was previously assigned a value, expression or parameter and the variable value needs to be calibrated, select the value you want to replace with a DCM\_CALIBRATE statement, e.g. 'delay=2.5n;' becomes 'delay=DCM\_CALIBRATE(test,result);'

**2.** Right-click and choose *Add DCM\_CALIBRATE*.

The DCM\_CALIBRATE form appears, displaying values derived from current tests, measures and results.



3. Choose a test from the *Test* dropdown list.
4. Choose a measure from the *Measure* dropdown list.
5. Choose a result from the *Result* dropdown list.
6. If you want to verify the measured result, select the *Verify* check box.

If you select this option, the measure / result, when used to calibrate this model, is then included in the spec sheet used to verify the model measurement against the design measurement.

7. Enter a value for the absolute tolerance in the Absolute Tolerance field.

Specify this value as the limit of the difference between the measured value from the design and the measured value from the model. So if you specify a value of 0.5, and the design measures 0.3, the model will fail if it is measured below -0.2 or above 0.8; otherwise it will pass.

8. Click *OK* to dismiss the form and save your changes.

The DCM\_CALIBRATE statement is created in the model.

## Modifying a Model Calibration Statement

You can modify a DCM\_CALIBRATE statement in the model:

1. In the model, select the entire line containing the DCM\_CALIBRATE statement.
2. Right-click and choose *Edit*.

The DCM\_CALIBRATE form appears, displaying the values for the selected DCM\_CALIBRATE statement.



3. Modify the values as desired.

For more information, see [Adding a Model Calibration Statement](#).

## **Virtuoso Analog Design Environment GXL User Guide**

### Cell Type Development

---

4. Click *OK* to dismiss the form and save your changes.

The DCM\_CALIBRATE statement is modified in the model.

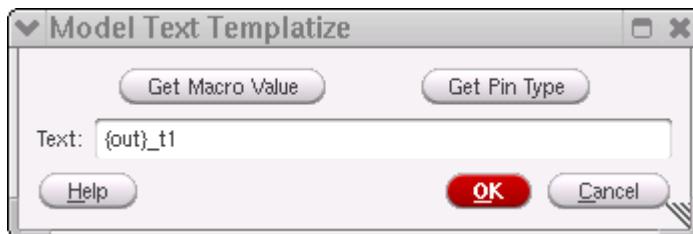
## **Templating Text**

You can select text in a model, then replace it with parameters or macro values or a pin type.

To template model text:

1. Select the text in the model that you want to template.
2. Right-click and choose *Edit*.

The Model Text Templatize form appears.



3. Do one of the following:

- Click *Get Macro Value* to replace the text with parameters or macro values.
- Click *Get Pin Type* to replace the text with a pin type.

## Specifying Model Options

You can modify the options for any model. These options include model behaviors, model options, options for calibration and verification, and advanced options such as dynamic Vdd sampling, measure and model output voltages, etc.



*Important*

Not all model options are applicable to all languages.

### Modifying Verilog-A Model Options

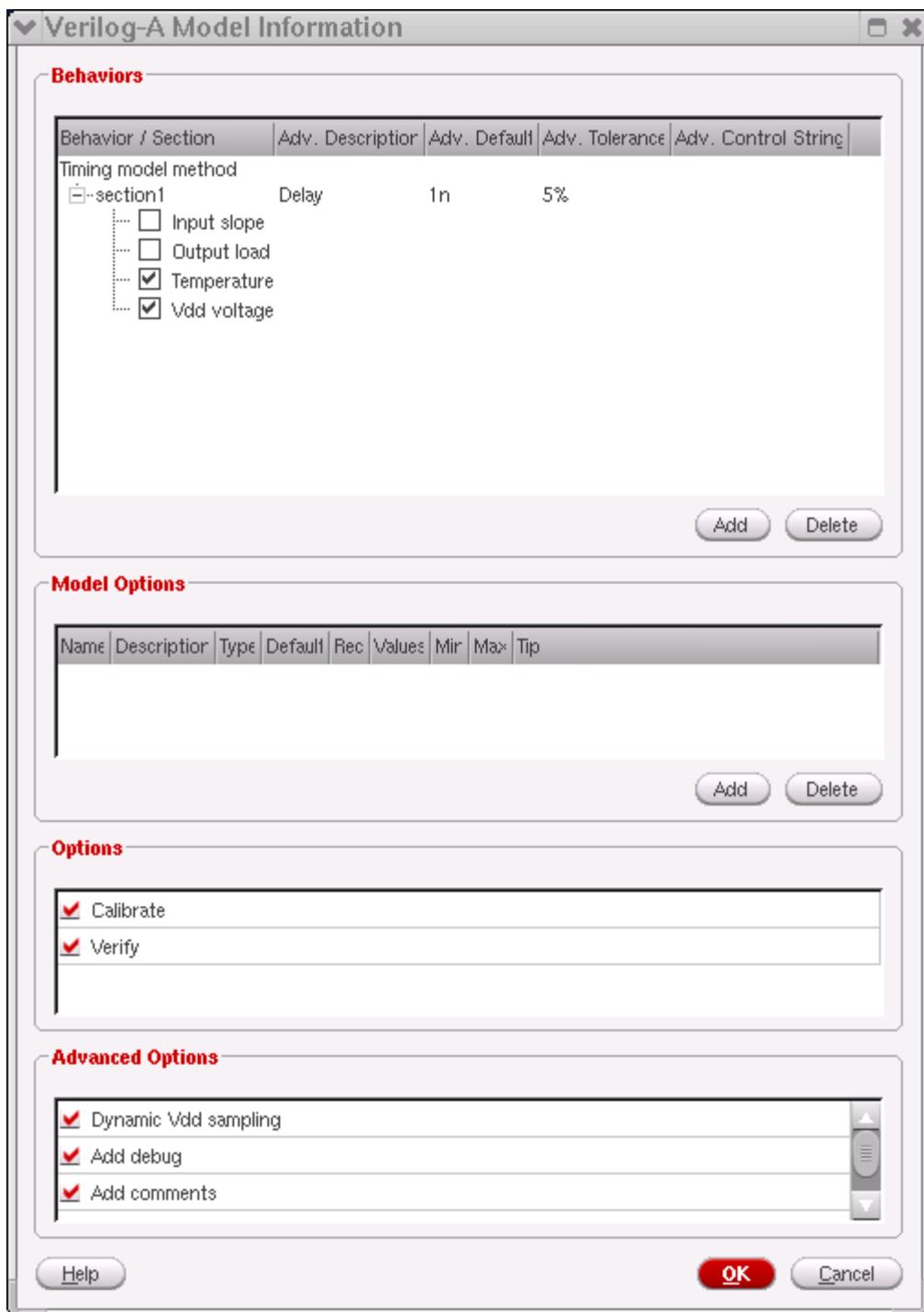
To modify Verilog-A model options:

- Click *Options* on the Verilog-A model tab.

# Virtuoso Analog Design Environment GXL User Guide

## Cell Type Development

The Model Information form appears.



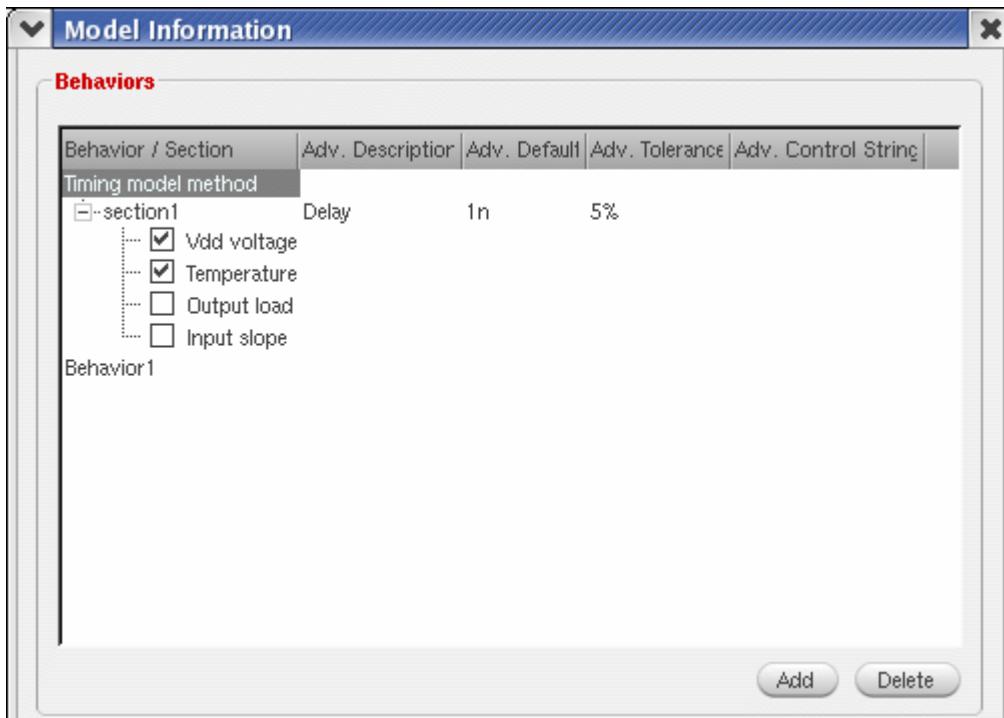
### ***Adding Model Behavior Options***

You can define a behavior to enable different swept parameters. A behavior also groups together one or more test benches using the DCM\_SECTION macro within the test. You can define multiple behaviors if some swept parameters do not apply to some measurements. For example, if you have timing and input capacitance tests, timing would support sweeping of Load,Slope, Temperature and Voltage, but input capacitance would normally have only Temperature and Voltage.

To add model behavior options:

1. On the Model Information form, click *Add*.  
A new behavior is listed in the Behaviors box.
2. Select, then select again the new Behavior1 field and enter an appropriate name. Press *Enter* when finished.
3. Quickly double-click the new behavior name to view the sections and model behavior options.
4. Select the check box for any declared parameters that you want to be made available for sweeping.

5. Double-click each field for the new behavior row, entering an appropriate value then pressing *Enter* when finished.



6. Repeat step 1 through step 5 for all model behaviors you want to add.

### ***Specifying Model Option***

Add a model option to add information required from the cell type user in order to create or calibrate the model. Within the template, this is accessed with the DCM\_VA\_VALUE macro.

To specify model options:

To add model behavior options:

1. On the Model Information form, click *Add*.

A new model option row is listed in the Model Options box.

2. Double-click each field for the new model option row, entering an appropriate value then pressing *Enter* when finished.
3. Repeat step 1 through step 5 for all model behaviors you want to add.

### ***Specifying Advanced Model Options***

To specify advanced model options:

1. In the Advanced Options box, select the *Dynamic Vdd sampling* check box if you want the model generated with dynamic vdd sampling.

If this option is enabled, VDD is dynamically sampled every time the model is evaluated (every time step). If this option is disabled, VDD is sampled once at the start of each simulation in the initial block of the model. For more information, see [Top-Down Verilog-A Block](#).

2. Select the *Add debug* check box to add debugging information when generating the model.



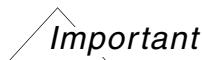
This option assumes that the DCM\_DEBUG macro has been used in the templates.

3. Select the *Add comments* check box to add comments when generating the model.



This option assumes that the DCM\_COMMENT macro has been used in the templates.

4. Select the *Measure and model output voltages* check box to generate measure and model output voltages with the model.



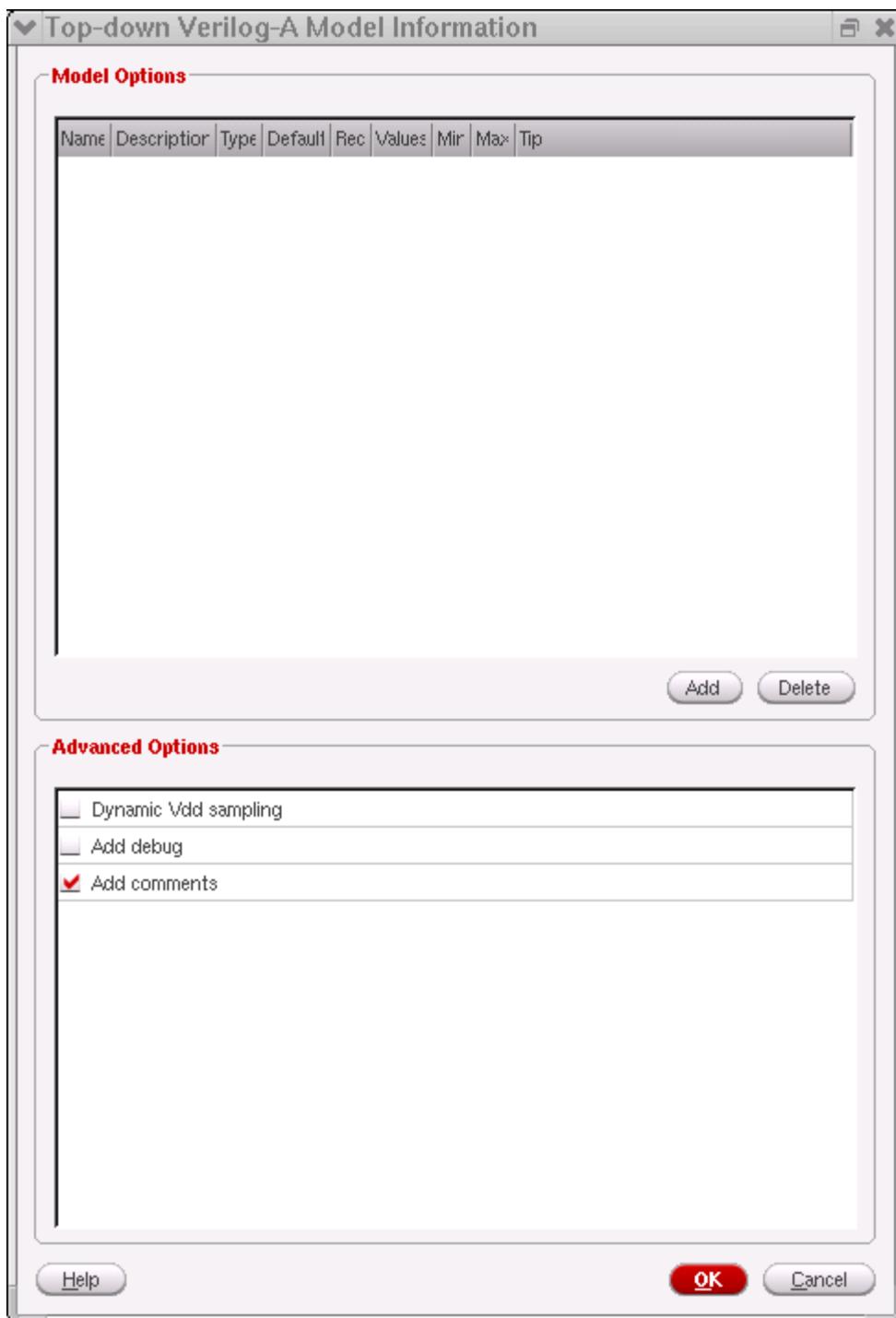
Enable this option only if the template has constructs that support it. For more information, see the [Condition Type](#) drop-down list in the Condition form.

### **Modifying Top-Down Verilog-A Model Options**

To modify top-down Verilog-A model options:

- Click *Options* on the Top-Down Verilog-A model tab.

The Model Information form appears.



### ***Specifying Model Option***

Add a model option to add information required from the cell type user in order to create or calibrate the model. Within the template, this is accessed with the `DCM_VA_VALUE` macro.

To specify model options:

To add model behavior options:

1. On the Model Information form, click *Add*.  
A new model option row is listed in the Model Options box.
2. Double-click each field for the new model option row, entering an appropriate value then pressing *Enter* when finished.
3. Repeat step 1 through step 5 for all model behaviors you want to add.

### ***Specifying Advanced Model Options***

To specify advanced model options:

1. In the Advanced Options box, select the *Dynamic Vdd sampling* check box if you want the model generated with dynamic vdd sampling.  
If this option is enabled, VDD is dynamically sampled every time the model is evaluated (every time step). If this option is disabled, VDD is sampled once at the start of each simulation in the initial block of the model. For more information, see [Top-Down Verilog-A Block](#).
2. Select the *Add debug* check box to add debugging information when generating the model.



This option assumes that the `DCM_DEBUG` macro has been used in the templates.

3. Select the *Add comments* check box to add comments when generating the model.



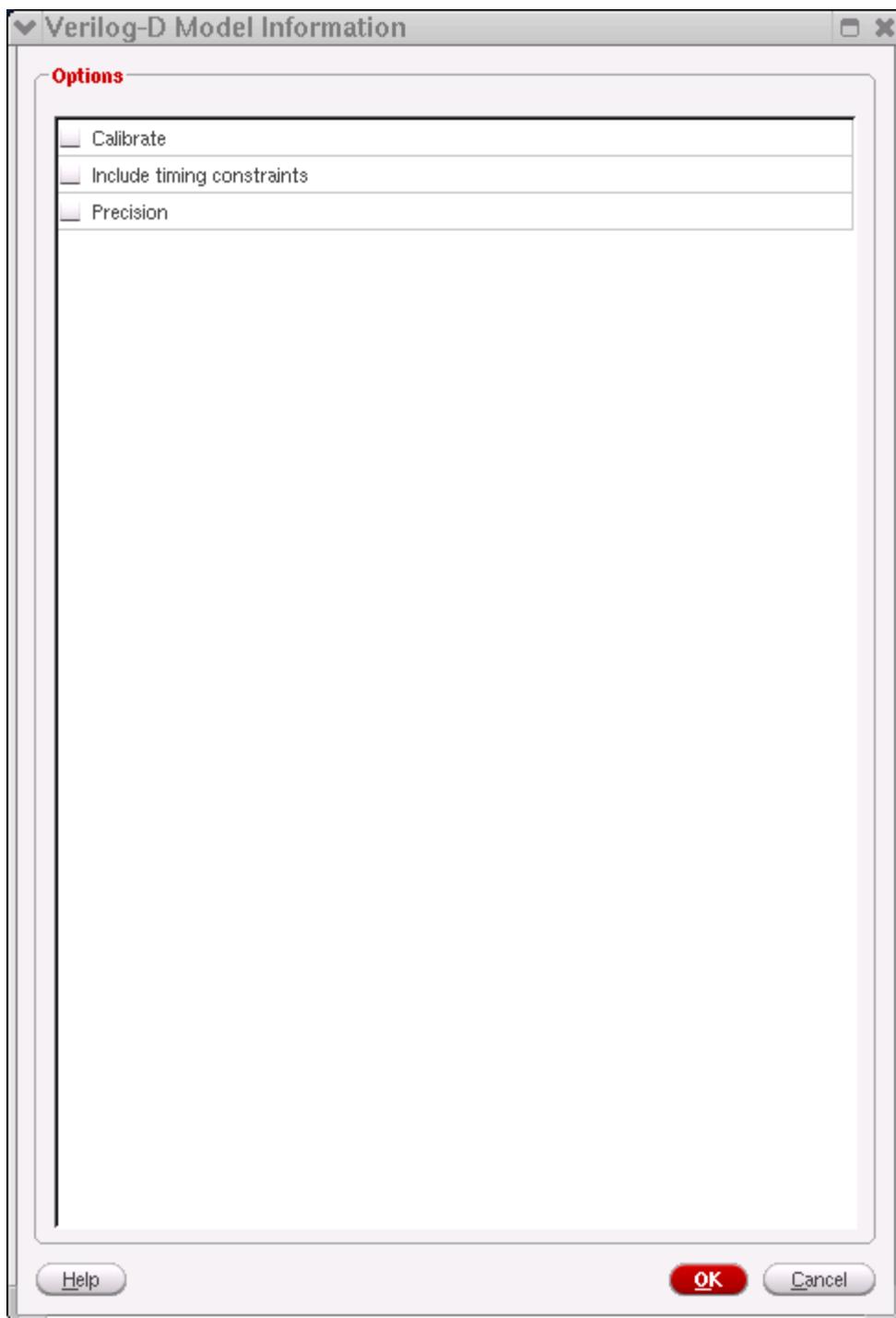
This option assumes that the `DCM_COMMENT` macro has been used in the templates.

## **Modifying Verilog-D Model Options**

To modify Verilog-D model options:

- Click *Options* on the Verilog-D model tab.

The Model Information form appears.



### ***Specifying Options***

The options are switches that enable or disable standard options that predefined by DCM and presented to the cel type user. These include model calibration and verification options and other advanced options.

To specify options:

1. In the Options box, select the *Calibrate* check box to calibrate this model.
2. Select the *Include timing constraints* check box if you want to include timing constraints.
3. Select the *Precision* check box to enable the Verilog-D precision option.

### ***Dismissing the Model Options Form***

When you are finished defining model options and behaviors:

- Click *OK*.

## **Specifying Cell Type Generation Options**

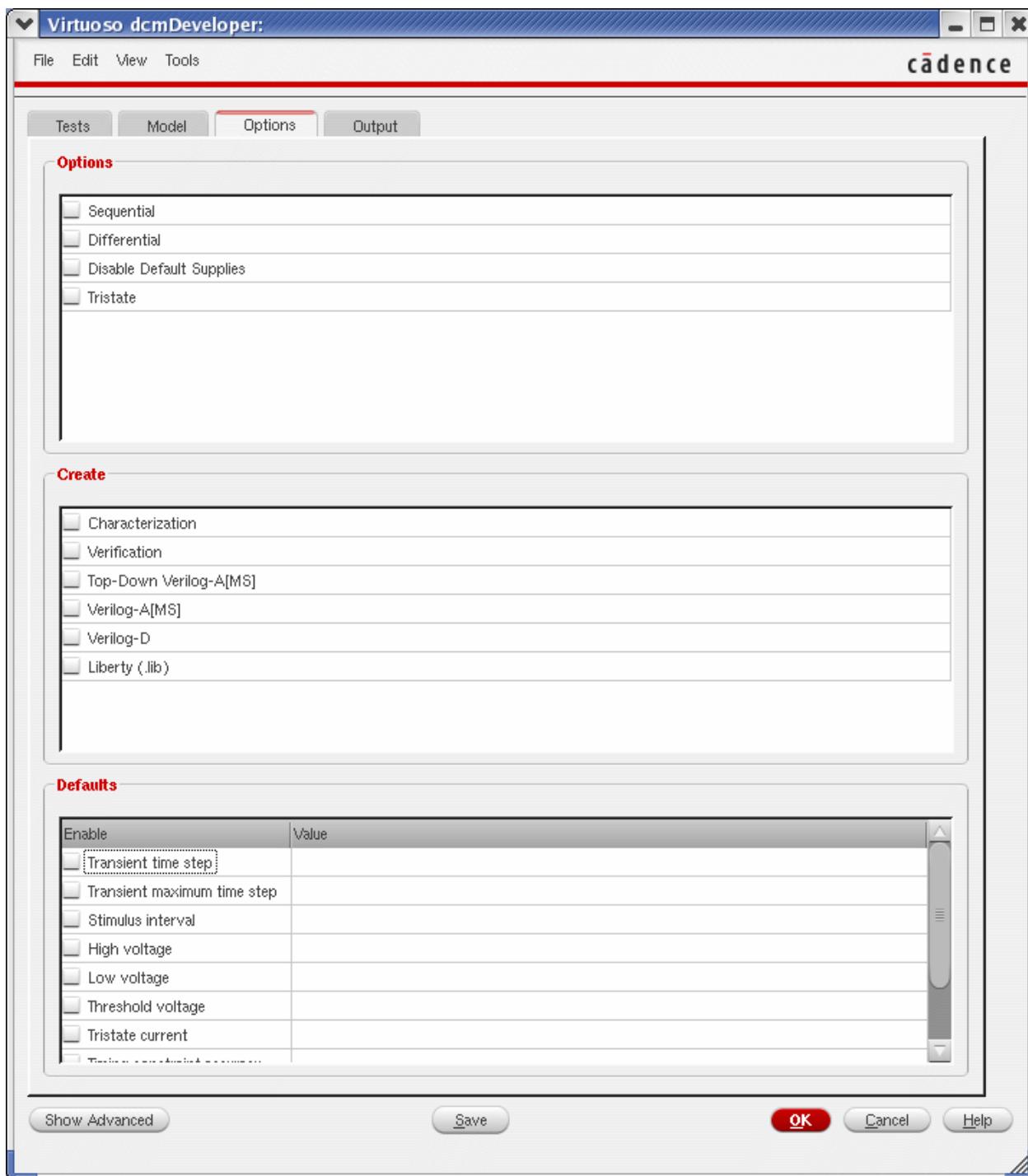
To begin setting options:

- Select the Options tab.

# Virtuoso Analog Design Environment GXL User Guide

## Cell Type Development

The Options page appears.



## Setting Cell Type Information

In the Options box, you provide information about the cell type generated. These options are used in conjunction with other information to configure the DCM interface for this cell type.



***The templates must include code that handles these behaviors depending on the switch setting.***

1. If the cell type can support differential pins, select the *Differential* check box.
2. If the cell type can support sequential, select the *Sequential* check box.
3. To disable default supplies, select the *Disable Default Supplies* check box.
4. If the cell type can support TriState, select the *TriState* check box.

## Specifying Cell Creation Options

You can specify options for cell creation in the Create box. For bottom-up modeling, if tests are provided, the model format and characterization options should be enabled.

1. To create characterization for the cell, click the Characterization check box.
2. To create verification in the cell, click the Verification check box.
3. To create Top-Down Verilog-A[MS] models, click the Top-Down Verilog-A[MS] check box.
4. To create Verilog-A[MS] models, click the Verilog-A[MS] check box.
5. To create Verilog-D models, click the Verilog-D check box.
6. To create Liberty models, click the Liberty (.lib) check box.

## Specifying Defaults

Defaults may be enabled and default values entered. These defaults then override the internal DCM defaults. Enabling these options make them available to the cell type user.



The templates should refer to the defaults.

1. To specify a default value for transient time step, select the Transient time step check box and enter a value in the Value field.
2. To specify a default value for transient maximum time step, select the Transient maximum time step check box and enter a value in the Value field.
3. To specify a default value for stimulus interval, select the Stimulus interval step check box and enter a value in the Value field.
4. To specify a default value for high voltage, select the High voltage check box and enter a value in the Value field.
5. To specify a default value for low voltage, select the Low voltage check box and enter a value in the Value field.
6. To specify a default value for threshold voltage, select the Threshold voltage check box and enter a value in the Value field.
7. To specify a default value for tristate current, select the Tristate current check box and enter a value in the Value field.
8. To specify a default value for timing constraint accuracy, select the Timing constraint accuracy check box and enter a value in the Value field.
9. To specify a default value for timing constraint delay, select the Timing constraint delay check box and enter a value in the Value field.
10. To specify a default value for stimulus repeat, select the Stimulus repeat check box and enter a value in the Value field.

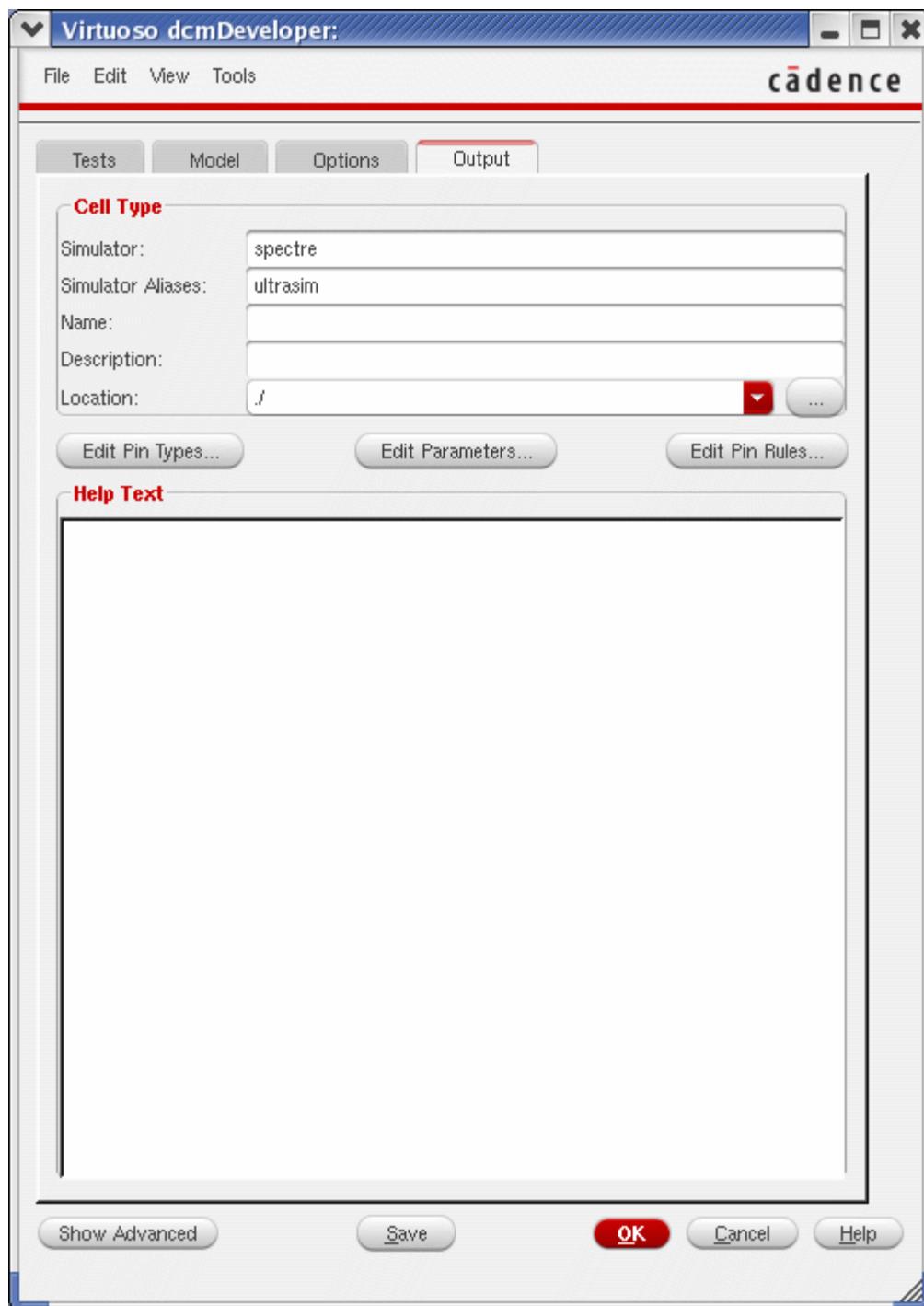
## **Specifying the Output Destination**

Specify the destination for the dcmDeveloper output on the Output page. You also define the primary simulator and aliases for any other simulators that can utilize this cell type and edit pin types, parameters and pin type rules. Finally, you can enter and modify help text that is then show in DCM when utilizing this cell type.

To begin setting output information:

1. Select the Output tab.

The Output page appears.



2. Enter the primary simulator name in the Simulator field.
3. Enter any other simulators that can use this cell type in the Simulator Aliases field.

4. Enter the cell name in the Name field.
5. Enter a description for the cell in the Description field.
6. Enter the path and filename for the cell in the Location field, or click ... to browse.
7. Add, modify, or delete pin types, parameters, and pin rules as desired.

For more information, see [Adding and Modifying Pin Types](#), [Adding and Modifying Pin Rules](#), and [Adding and Modifying Parameters](#).

8. Enter any helpful text you want displayed in the output in the Help Text box.
9. Click *OK* to check and save the current information, then dismiss the form.

## Adding and Modifying Pin Rules

You can add, delete, and modify pin rules from the Output page. Pin type rules enable you to add error checks for a number of design pins that can have a given type.

For more information, see:

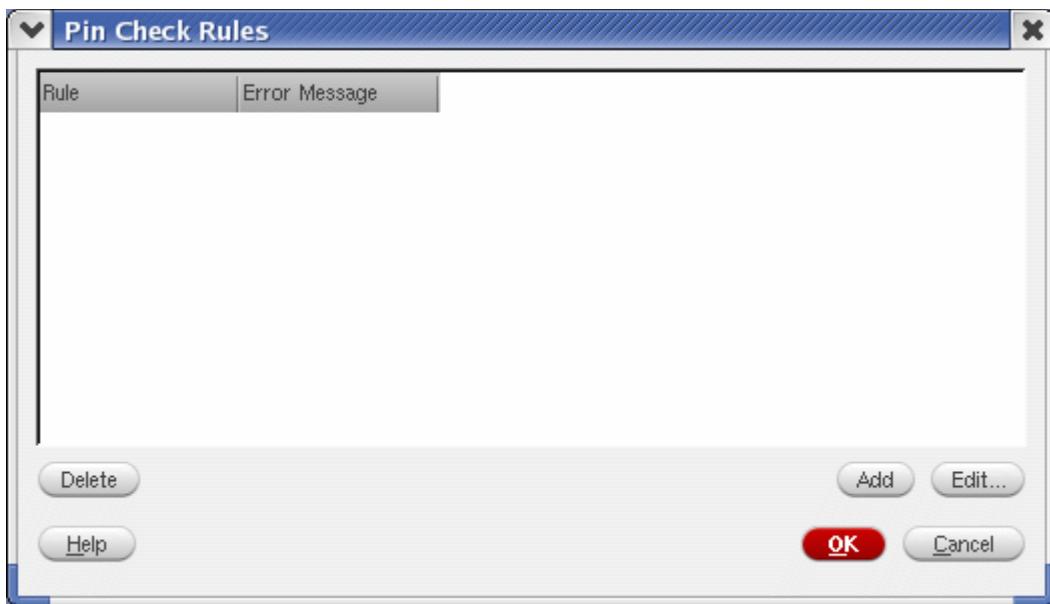
- [Adding Pin Rules](#)
- [Modifying Pin Rules](#)
- [Deleting a Pin Rule](#)

### Adding Pin Rules

To add a new pin rule:

1. Choose *Tools — Edit Pin Type Rules*.

The Pin Check Rules form appears.



2. Click *Add*.

A new, empty row appears in the Pin Check Rules form and the Edit Pin Type Check Rules form appears.



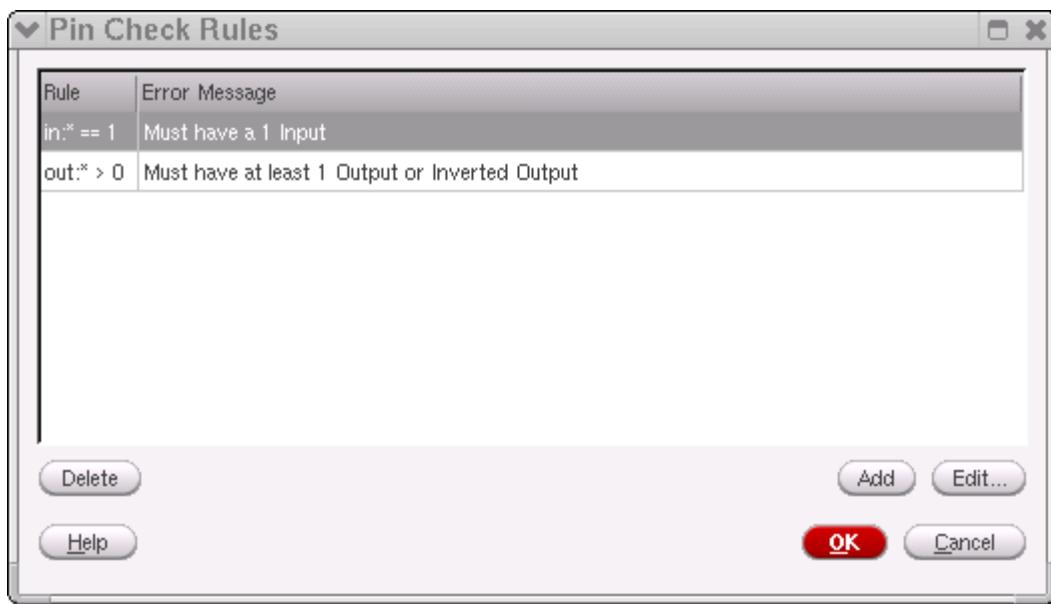
3. Choose a pin type from the *Pin Type* drop-down list.
4. Choose a requirement for the pin type from the *Require* drop-down list.
5. In the Error field, specify an error message for the case where this rule is not met.
6. Under *Pin Type Attribute*, select a radio button to indicate whether the rule applies to all pins, inverted pins only, or non-inverted pins only.
7. If you chose a complex check from the Require drop-down, specify the modifier and number of design pins for the check under *Complex Check*.
8. Click *OK* to save your changes and dismiss the Edit Pin Type Check Rules form.  
Note that the new check appears in the Pin Check Rules form.
9. Repeat step 2 through step 8 for all rules you want to add.
10. Click *OK* to save your changes and dismiss the Pin Check Rules form.

## Modifying Pin Rules

To edit pin rules:

1. Choose *Tools — Edit Pin Type Rules*.

The Pin Check Rules form appears.



2. Select the rule you want to modify.

3. Click *Edit*.

The Edit Pin Type Check Rules form appears, displaying the selected rule.



4. Modify the fields as desired.

For more information on this form, see [Adding Pin Rules](#).

5. Click *OK* to save your changes and dismiss the Edit Pin Type Check Rules form.

Note the changes to the rule in the Pin Check Rules form.

6. Click *OK* to save your changes and dismiss the Pin Check Rules form.

### **Deleting a Pin Rule**

To delete pin rules:

1. Choose *Tools — Edit Pin Type Rules*.  
The Pin Check Rules form appears.
2. Select the row for the pin rule you want to delete.
3. Click *Delete*.
4. Click *OK* to dismiss the form.

## Saving the dcmDeveloper Data

To save your data:

- Click *Save*.

To save data to a different cell type name:

1. Choose *File — Save As*.  
The Save Cell Type form appears.
2. Specify a path and cell type name.
3. Click *OK*.

## Closing dcmDeveloper

To close dcmDeveloper and save your data:

- Click *OK*.

To close dcmDeveloper without saving data, do one of the following:

- Click *Cancel*.
- Choose *File — Quit*.

## Viewing an Existing Cell Type

You can examine an existing cell type in dcmDeveloper.

You can open an existing cell type either from within the [main DCM GUI](#), from [dcmDeveloper](#), or from the [Cell Type form](#).

### Opening from DCM

To open an existing cell type from within the main DCM gui:

- Choose *Tools — Edit Selected Function* to launch dcmDeveloper and load the cell type for the design specified.

The dcmDeveloper window appears.

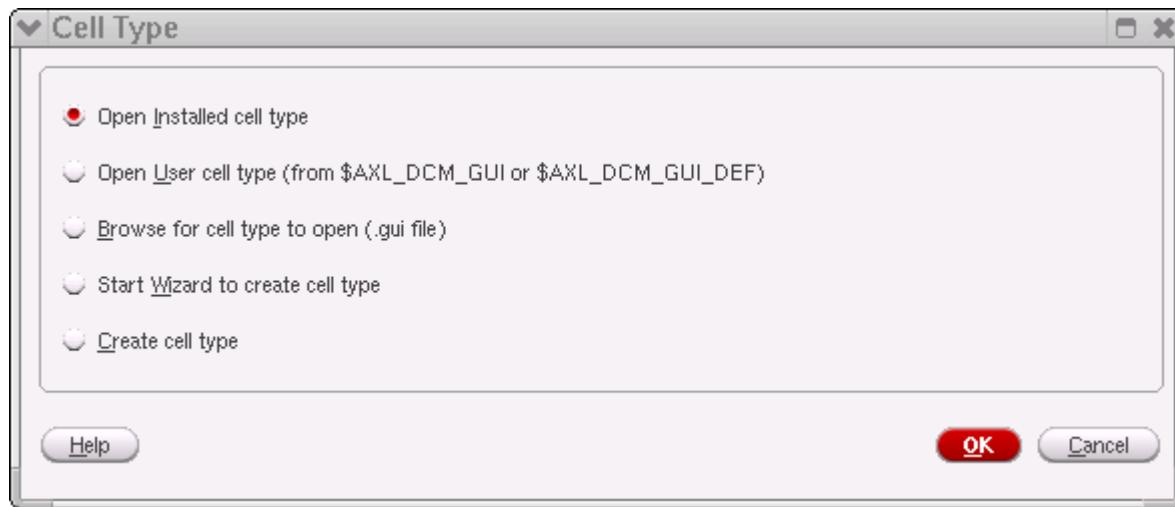
For more information on the different panels in dcmDeveloper, see [Understanding the dcmDeveloper Form](#).

### Opening from dcmDeveloper

To open from within dcmDeveloper:

1. View the Cell Type form.

The Cell Type form is displayed when you launch dcmDeveloper. You can also choose *File — Open Cell Type*.



2. Select one of the following radio buttons:

- Open Installed cell type
  - Open User cell type
  - Browse for cell type to open
  - [Start Wizard to create cell type](#)
  - [Create cell type](#)
- 3.** Click *OK*.

The specified cell type loads in dcmDeveloper.

For more information on the different panels in dcmDeveloper, see [Understanding the dcmDeveloper Form](#).

# **Virtuoso Analog Design Environment GXL User Guide**

## Cell Type Development

---

---

## OpenDCM

---

OpenDCM provides a mechanism for adding functional cell types to the Design Characterization and Modeling (DCM) feature of the Virtuoso® Analog Design Environment GXL. You can specify one or more directories containing OpenDCM cell types using the [AXL DCM GUI](#) or [AXL DCM GUI DEF](#) environment variable.

See following topics for more information:

- [OpenDCM File Types](#) on page 430
- [OpenDCM Directory Structure](#) on page 431
- [OpenDCM .gui File](#) on page 433
- [OpenDCM Test Template](#) on page 455
- [OpenDCM Model Template](#) on page 468
- [OpenDCM .txt Help File](#) on page 476
- [OpenDCM Pin Type Substitutions](#) on page 477
- [OpenDCM Functions](#) on page 479
- [OpenDCM Variables](#) on page 508
- [OpenDCM Command and Macro Reference](#) on page 508
- [Creating Templates That Support Differential Pins](#) on page 511

## OpenDCM File Types

You can create the following set of files to describe each functional cell type:

Extension	Description
.gui	Setup file defining pin types, parameters, options, models, and sweeps that are supported by the OpenDCM model and test templates  <b>Note:</b> The .gui files for the set of functional cell types provided with DCM are installed in <i>your_install_dir/tools/dfII/etc/dcm/cellTypes</i> in subdirectories that match the categories that appear in the <i>Function</i> list box on the <i>Function</i> tab of the Virtuoso Design Characterization and Modeling window.
.ade	OpenDCM test template file containing the simulation netlist and ADE state information
.va	<u>OpenDCM model template</u> file containing conditional statements used to generate Verilog-A[MS] (.va), Verilog-D (.v), or Liberty (.lib) code based on the pin combinations and characterization options you specified
.v	
.lib	
.txt	Optional file that contains help text that appears when you select a function and choose <i>View – Function Help</i> (see “ <u>OpenDCM .txt Help File</u> ” on page 476)

The file extension of the template indicates the model type as follows:

### Extension Model Template Type

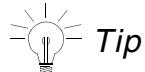
.va	Verilog-A or Verilog-AMS, where the file name for a top-down model is always <i>top_down.va</i>
.v	Verilog-D
.lib	Liberty

The .gui file contains sections for

- common information (such as *function* and *parameters*),
- model generation information (such as *Verilog-A* and *Sweeps*), and

- design verification information (in the `Tests` block).

See also “[OpenDCM Directory Structure](#)” on page 431.



OpenDCM writes the results of each intermediate processing step to a sequentially-indexed file in the `./dcm/dcmName/odcm` directory. For debugging purposes, it can be helpful to look in the last file that OpenDCM processed before a problem occurred.

## OpenDCM Directory Structure

The directory structure for OpenDCM files is as follows:

- The `.gui` files are located in the directories you declare using the [AXL\\_DCM\\_GUI](#) or [AXL\\_DCM\\_GUI\\_DEF](#) environment variable.
- The `.lib` (Liberty), `.v` (Verilog-D), `.va` (Verilog-A[MS]), and `.ade` (test) files are located down two directories from the `.gui` file, in `guiFileName/simulatorName` (that is, a directory within a directory where the first level is the name of the `.gui` file and the next level is the simulator name)
- The optional `.txt` file is located in the same directory as the `.gui` file.

For example, consider a `Samples` directory declared as follows:

```
setenv AXL_DCM_GUI $PROJECT1/samples/dcm/odcm/Samples...
```

**Note:** `Samples` appears as a choice in the *Function* list box.

The `Samples` directory might contain the following `.gui` and `.txt` files (for example):

```
o_myand.gui      o_myand.txt
o_mychp.gui     o_mychp.txt
o_mypfd.gui    o_mypfd.txt
o_myvco.gui    o_myvco.txt
```

For each cell type, there is a subdirectory that matches the name of the `.gui` file.

For example:

```
o_myand
o_mychp
o_mypfd
o_myvco
```

Each cell-type directory contains a subdirectory whose name matches the target simulator name (for characterization) for the OpenDCM templates. The `targetSimulator` subdirectory contains the OpenDCM test and model template files (`.ade`, `.va`).

## Virtuoso Analog Design Environment GXL User Guide

### OpenDCM

---

For example:

Directory	Description
Samples	<i>Function</i> list box text
o_myand.gui	Required .gui file
o_myand.txt	Optional .txt file
o_myand	Cell-type subdirectory whose name matches the .gui file name <sup>1</sup>
spectre	Target characterization simulator subdirectory <sup>2</sup>
and.va	Verilog-A model template file
timing.ade	OpenDCM test template file

1. This name appears as a subcategory under *Samples* in the *Function* list box
2. This name must match the name specified in the odcm block of the .gui file

## OpenDCM .gui File

Each OpenDCM cell type is defined in a `.gui` file. One or more directories containing `.gui` files is declared using the `AXL_DCM_GUI` or `AXL_DCM_GUI_DEF` environment variable. Each `.gui` file contains blocks of information using the following structure:

```
blockName {  
blockStatements  
}
```

where each `blockName` is one of the following and `blockStatements` is one or more valid statements for that block:

<b><code>blockName</code></b>	<b>Description</b>
<code>function{...}</code>	Provides text for the <i>Function</i> list box, defines pin types for the cell and conditional statements for those pins
<code>odcm{...}</code>	Specifies one or more target simulators (for characterization) supported by the OpenDCM templates
 <b>Important</b>	
	All <code>.gui</code> file must contain an <code>odcm</code> block.
<code>create{...}</code>	Specifies the items that can be created for this cell type
<code>parameters{...}</code>	Defines cell-specific parameters that will appear in the <i>Parameters</i> group box on the <i>Parameters</i> tab of the Virtuoso Design Characterization and Modeling window
<code>Top-Down Verilog-A{...}</code>	For model generation, defines items that will appear on the <i>Top-Down</i> tab for Verilog-A[MS] modeling
	<b>Note:</b> If you do not provide a <code>Top-Down Verilog-A</code> block, the program will derive top-down Verilog-A information from the <code>Verilog-A</code> block, if one exists.
<code>Verilog-A{...}</code>	For model generation, defines items that will appear on the <i>Verilog-A[MS]</i> tab
<code>Verilog-D{...}</code>	For model generation, defines items that will appear in the <i>Options</i> group box on the <i>Verilog-D</i> tab
<code>Liberty{...}</code>	For model generation, defines items that will appear in the <i>Options</i> group box on the <i>Liberty</i> tab (content currently ignored)

<b>blockName</b>	<b>Description</b>
<u>Defaults</u> { ... }	Defines options that will appear on the <i>Parameters</i> tab of the Virtuoso Design Characterization and Modeling window
<u>Sweeps</u> { ... }	For model generation, specifies default sweep values that will appear on the <i>Sweeps</i> tab of the Virtuoso Design Characterization and Modeling window

Details for each block are provided in the following sections.

## function Block

The function block of the .gui file contains the following information:

```
function {
    "functionName"
    pinTypes {
        pinStatements
    }
    pinFields {
        rowLabel
    }
    check {
        pinRules
    }
}
```

and optionally:

```
    options {
        functionOptions
    }
}
```

Items specified in the function block include:

*functionName*      Function name that appears in the *Function* list area on the *Function* tab



This text must be unique for all functions in all categories in all .gui files from all cell-type directories.

*pinStatements*      One or more pin type definitions

<i>rowLabel</i>	Label text for an optional row that appears on the <a href="#">Design Pin Attributes</a> form when you click <i>Attributes</i> for the pin in the <i>Design pin types</i> table on the <i>Function</i> tab
	<b>Note:</b> The program processes the <i>pinFields</i> section only when your <i>pinStatements</i> include the <i>pinfield pinExtra</i> .
<i>pinRules</i>	One or more rules for validating the set of pins selected on the <i>Function</i> tab of the Virtuoso Design Characterization and Modeling window
<i>functionOptions</i>	One or more function options

For example:

```
function {
    "Voltage-Controlled Oscillator"
    pinTypes {
        in      "Control"           def=(co.*|vi.*|ct.*)   dir=in   entry=low
        reset:i "Low Reset"        def=(r.*|c.*).*(n|z)   dir=in
        reset   "High Reset"       def=(r.*|c.*)
        ibias   "Ibias"            def=(i.*|b.*)
        out:1   "Primary Output"  def=o.*
        out     "Output"           def=o.*                 dir=out
    }
    check {
        in:*= 1      "Must have one Control input"
        reset:*< 2   "Must have a maximum of 1 Reset input"
        ibias:*< 2   "Must have a maximum of 1 Ibias input"
        out:*> 0     "Must have at least 1 Output"
    }
}
```

## Pin Type Definitions

OpenDCM supports the definition of pin types, rather than pins, so that pin names can be specified by the user of a functional cell type in DCM. A pin type can be optional such that, for a single functional cell type, multiple variations of a device can be supported (for example, a VCO that has a single reset pin but multiple output pins). See also [“Pin Check Statements”](#) on page 440.

Pin type definitions appear in the *pinTypes* section of the *function* block of the *.gui* file:

```
pinTypes {
    pinStatements
}
```

where *pinStatements* is one or more pin definitions specified as follows:

```
pinType[:pinAttribute] "pinDescription" [pinExtra-]
```

Arguments of the *pinStatement* are as follows:

- *pinType* is the pin type used in OpenDCM templates; *pinType* can contain any alphanumeric and the underscore character.

**Note:** The program adds pin name `unused` automatically for all cell types; the program also adds pin names `vdd` and `vss` automatically for all cell types unless you specify `disable_default_supplies` in the options section of the function block.
  - *pinAttribute* is an optional alphanumeric token used to identify a group of pins that have a behavior in common; for example, one of the following pre-defined attribute names:
    - `i` — indicates an inverted pin type
    - `n` — indicates a non-inverted pin type

**Note:** This pin attribute is ignored.

    - `1` — indicates the first in a list of pins of type *pinType*

**Note:** The `DCM_GENERATE` and `DCM_FOREACH` functions can be used to iterate through all pins of a particular *pinType* in the templates, and generate different code based on the *pinAttribute* of a pin—for example, see “[DCM\\_INV](#)” on page 481.
  - *pinDescription* is a text string that describes the *pinType*; *pinDescription* can contain spaces.
- Note:** The set of pin descriptions for a cell type appears on the drop-down list in the *Pin Type* column for each *Pin Name* in the table on the *Function* tab of the Virtuoso Design Characterization and Modeling window. The program adds pin name `unused` automatically for all cell types; the program also adds pin names `vdd` and `vss` automatically for all cell types unless you specify `disable_default_supplies` in the options section of the function block.

- *pinExtra* is zero or more of the following optional specifications used to provide additional information about each pin, as well as to define any changes to the Virtuoso Design Characterization and Modeling window when you select the pin type:

<b><i>pinExtra</i></b>	<b>Description</b>
<code>diff=diffAllowed</code>	<p>Differential pin setup check (performed on save): If <code>diff=no</code> for this pin and the <i>Differential</i> <u>check box</u> is marked in the <i>Design pin types</i> group box on the <i>Functions</i> tab form, the program verifies that pins of this type have no differential setups and issues an error message if this condition has been violated</p> <p>If you do not specify <code>diff pinExtra</code>, or if it holds any other value besides <code>no</code>, the program does not perform this check.</p> <p><b>Note:</b> The program also supports <code>cml=diffAllowed</code> because <u>ModelWriter</u> uses this syntax (instead of <code>diff=diffAllowed</code>).</p>
<code>def=regPerlExpr</code>	Condition definition, where <code>regPerlExpr</code> is a regular Perl expression defining a condition for assigning this <i>pinType</i> as the default for a particular design pin
<code>dir=dirSpec</code>	<p>Pin direction, where <code>dirSpec</code> is one of the following tokens specifying the direction of a pin:</p> <ul style="list-style-type: none"> <li>■ <code>in</code></li> <li>■ <code>out</code></li> <li>■ <code>inout</code></li> </ul> <p><b>Note:</b> Direction information is used during characterization and modeling. <code>in</code> pins affect input capacitance; <code>out</code> pins affect load dependency; <code>inout</code> pins are bidirectional and used for pads.</p>
<code>dis=pinDiscipline</code>	<p>Pin discipline, where <code>pinDiscipline</code> is a valid Verilog-A or Verilog-AMS discipline for the model template, or <code>none</code> to disable discipline creation for design pins of this type (something you might want to do for AMS digital pins); you can also specify <code>dis=reg</code> for AMS digital output pins that hold values</p> <p><b>Note:</b> The default <i>pinDiscipline</i> is electrical.</p>

<b><i>pinExtra</i></b>	<b>Description</b>
<code>entry=entrySpec</code>	Valid entries, where <code>entrySpec</code> is one of the following tokens: <ul style="list-style-type: none"> <li>■ low—only the <i>Low Voltage</i> value can be specified</li> <li>■ high—only the <i>High Voltage</i> value can be specified</li> <li>■ none—neither value can be specified</li> <li>■ both—both values can be specified</li> </ul>
<code>pinfield=pinFieldAttr</code>	Pin field enable, where <code>pinFieldAttr</code> is typically a one-letter name for the attribute token to be used in the <code>DCM_PIN_ATT</code> function
<code>tip="tipText"</code>	Tip text for the pin

For example:

```
pinTypes {
in "Control" def=(co.*|vi.*|ct.*) dir=in entry=low tip="Control voltage
input"
reset:i "Low Reset" def=(r.*|c.*).*(n|z) dir=in tip="When low, Output is low"
reset "High Reset" def=(r.*|c.*).*(n|z) dir=in tip="When high, Output is low"
ibias "Ibias" def=(i.*|b.*).*(n|z) dir=in entry=low
out:l "Primary Output" def=o.* dir=out tip="First oscillator output"
out "Output" def=o.* dir=out tip="Oscillator output"
}
```

## Pin Field Label



You must specify a `pinfield` `pinExtra` for a `pinType` before you can use the `pinFields` section. For example:

```
pinTypes {
...
# Divider output
```

```
out "Output" def=(o|y|q|z) dir=out tip="Resets low, presets high" pinfield=d  
}
```

Use the `pinFields` section of the function block of the `.gui` file to specify the label for the additional row that appears on the [Design Pin Attributes](#) form when you click *Attributes* for the pin in the *Design pin types* table on the *Function* tab as follows:

```
pinFields {  
    rowLabel  
}
```

Specify the `rowLabel` all on one line as follows:

```
"titleText" tip="tooltipText" rowLabelOptions
```

`titleText`                    Row label text

`tooltipText`                Tooltip text that appears

`rowLabelOptions` can be zero or more of the following:

`name=pinFieldAttr`      A string that matches the pin field attribute token you specified for the [pinfield pinExtra](#) in your [pin type definition](#)

**Note:** Recommended for associating a particular `pinFieldAttr` with the `pinFields` information, especially when you have more than one `pinFieldAttr` defined.

`req= { yes | no }`     (Optional) Flag to indicate whether you must specify a value for this attribute

yes                          You must specify a value

no                           You do not have to specify a value

Default Value: no

`type=type`              Type of value  
Valid Values:

int                          Value must be an integer

num                          Value must be a number

str                          Value can be any string

`min=minLimit`          Minimum valid value for the indicated type (`type=type`)

`max=maxLimit`          Maximum valid value for the indicated type (`type=type`)

For example:

```
pinFields {  
# Count for divider  
"Count" name=d tip="For divider outputs use numbers > 1. Use 1 for buffered/inverted  
clock" req=yes type=int min=1  
}
```

## Pin Check Statements

You can specify selection requirements for pins you defined in the `pinTypes` block as "rules" that appear in the `check` section of the `function` block of the `.gui` file using the following syntax:

```
check {  
    pinRules  
}
```

where `pinRules` is one or more rules for validating the set of pins selected on the *Function* tab of the Virtuoso Design Characterization and Modeling window specified as follows:

```
pinExpression "pinError"
```

where `pinExpression` is a regular expression involving pins declared in the `pinTypes` block, and `pinError` is the message that results if `pinExpression` does not evaluate to true. The `pinExpression` can contain wildcards (for example, `pinName:*`) as well as logical operators (for example, `<=`, `==`, `>=`, `&&`, `||`, `<`, `>`).

For example:

```
check {  
    in:* == 1      "Must have one Control input"  
    reset:* < 2   "Must have a maximum of 1 Reset input"  
    ibias:* < 2   "Must have a maximum of 1 Ibias input"  
    out:* > 0     "Must have at least 1 Output"  
    up == 1 || up:i == 1 "Must have 1 'Inverted Up Output' and/or 1 'Up Output'"  
    vdd < 1        "No Vdd pins allowed"  
    vss == 1        "Must have 1 Vss pin"  
    unused > 1     "Must have > 1 Unused pin"  
}
```

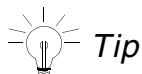
## Function Options

The options section of the function block of the .gui file, if specified, can contain one or more of the following function options:

<i>functionOption</i>	Description
diff	Enables the <i>Differential</i> check box in the <i>Design pin types</i> group box on the <u>Function</u> tab
disable_default_supplies	Disables the addition of the default <i>Vdd</i> and <i>Vss</i> selections to the <i>Pin Type</i> drop-down list in the <i>Design pin types</i> table on the <u>Function</u> tab
seq	Specifies that the cell type has stored states



***When default supplies are disabled, caution must be exercised when accessing high, low, and threshold values (see “Pin Value Access Functions” on page 482).***



You can append additional characters to the end of a *functionOption* keyword for readability. For example, you can specify seq as sequential.

For example:

```
options {
    sequential
    diff
}
```

## odcm Block



Your .gui file must contain an odcm block.

You specify the target simulators supported by OpenDCM templates for characterization in the `odcm` block of the `.gui` file as follows:

```
odcm {  
    targetSimulator[ = simulatorAliasList ]  
    ...  
}
```

where

- `targetSimulator` is the target simulator name (for characterization) and must match the simulator subdirectory name as it appears in the cell-type subdirectory.
- `simulatorAliasList` is a comma-separated list of one or more target simulator names. The list may contain any target simulator that the Analog Design Environment supports.

OpenDCM uses the templates from the `targetSimulator` directory.  
See also “[OpenDCM Directory Structure](#)” on page 431.

Here are some examples:

```
odcm {  
    spectre  
}  
  
odcm {  
    spectre = ultrasim  
}
```

## create Block

You can specify what you want DCM to create for a cell type in the `create` block of the `.gui` file as follows:

```
create {  
    createItems  
}
```

where `createItems` is one or more of the following items:

- `Characterization` to specify a characterization for model calibration
- `Verilog-A` (or `Verilog-AMS`) to specify a bottom-up Verilog-A (or Verilog-AMS) model
- `Verilog-D` to specify a bottom-up Verilog-D model
- `Liberty` to specify a bottom-up Liberty .lib model
- `Top-Down Verilog-A` (or `Top-Down Verilog-AMS`) to specify a top-down Verilog-A (or top-down Verilog-AMS) model

- Verification to specify design verification

For example:

```
create {
    Characterization
    Verilog-A
}
```

## parameters Block

You declare parameters for a cell type in the parameters block of the .gui file as follows:

```
parameters {
    parameterStatements
}
```

*parameterStatements* is one or more parameter declarations using the following syntax:

```
guiName parameterName "parameterDescription" [parameterExtras]
```

*guiName* Name you use to reference this parameter in the .gui and other template files.

*parameterName* Parameter name.  
The parameter name can contain references to pin types as defined in the *pinTypes* block such that the program substitutes the first pin of a referenced type where the token appears between curly braces in the *parameterName*.



You should use a prefix of `dcm_` for each *parameterName* so that the final parameter name is sure to be unique.

*parameterDescription*

Text that appears on the *Parameters* tab.

*parameterExtras* Zero or more of the following *name=value* pairs that provide additional information about a parameter:

*va=variableAccess*

Variable used to pass the value of a swept parameter into the table file containing measured results for the calibrated Verilog-A (or Verilog-AMS) model, where *variableAccess* is one of the following pre-defined real variables:

- *temp* for the temperature
- *vdd* for the supply voltage

Alternatively, *variableAccess* can be a Verilog-A[MS] signal access function<sup>1</sup>:

- *v(pinType)* signal access for voltage
- *I(pinType)* signal access for current

**Note:** When you use a signal access function, the first pin of type *pinType* gets substituted where the token appears between parentheses. You can multiply this function by negative one (\*-1) to convert a negative value, like bias current, to a positive one; for example, *va=I(ibias)\*-1*.

*def=defaultValue*

Default value for the parameter; for example, *def=3*

*units=parameterUnits*

Units for the parameter; for example, *units=Volts*

`type="type"`

Type identifier for Liberty model calibration

Valid Values:

- load
- slope
- temp
- voltage (or vdd)
- process

or for data verification

Valid Values:

- int – integer
- str – string
- num – number
- unk – unchecked, same as str

`tip="parameterTip"`

Tip text for the parameter that appears in the lower left corner of the status bar of the Virtuoso Design Characterization and Modeling window when the cursor first hovers over the parameter name, either in the *Parameters* group box on the Parameters tab or on the Sweeps tab

1. Use of these functions is only recommended for cases where DCM does not already sample the voltage or current. Also, Verilog-A models for Spectre circuit simulation cannot contain these functions within a ddt function (which is used to model input capacitance in DCM); so `variableAccess` must be a real variable, and the Verilog-A code that assigns the voltage or current to the real variable must be added to the Verilog-A model template.

For example:

```
parameters {
  in      dcm_{in}_value    "Ctl voltage" va=v(in)      def=3      units=Volts
  tip="Control voltage"
  from    dcm_{in}_from     "Input voltage from"      def=0      units=Volts
  tip="Start value of Control voltage" use=ver
  to      dcm_{in}_to       "Input voltage to"        def=3      units=Volts
  tip="End value of Control voltage" use=ver
  by      dcm_{in}_by       "Input voltage by"        def=0.5    units=Volts
  tip="Step value of Control voltage" use=ver}
```

```
vdd          dcm_{vdd}_value  "Vdd voltage"   va=vdd      def=3    units=Volts
temperature  dcm_temperature "Temperature"   va=temp     def=27   units="Degrees C"
slope        dcm_slope       "Input slope"    def=0.2n   units=Seconds
load_cap    dcm_load_cap   "Output load"   def=100f   units=Farads
ibias        dcm_{bias}_value "Bias current"  va=I(bias) def=-50u   units=Amps
}
```

**Note:** You can sweep any parameter you declare in the parameters block.

See also “[DCM\\_PARAM\\_VA](#)” on page 486 for information about associating Verilog-A code or parameter names with DCM parameters.

## Top-Down Verilog-A Block

You specify items that will appear on the *Top-Down* tab and on the Advanced Verilog-A form for Verilog-A[MS] modeling in the Top-Down Verilog-A block of the .gui file.

### Top-Down Verilog-A {

```
model {
    modelParameterDefinitions
}
```

Items that appear in the *Options* group box on the *Top-Down* tab

**Note:** You can use the [DCM\\_VA\\_VALUE](#) function in test and model templates to access *modelOption* values.

```
advanced {
    options {
        optionSpecifiers
    }
}
```

Items that appear on the [Advanced Verilog-A form](#) when you click *Advanced* on the *Top-Down Verilog-A[MS]* tab

Valid Values for *optionSpecifiers*:

vdd\_sample      *Dynamic Vdd sampling*

debug            *Add debug*

comments        *Add comments*

**Note:** The *Include file in header* and *Script output in header* data entry fields are always available on the form.

```
}
```

End of Top-Down Verilog-A block

For example:

```
Top-Down Verilog-A {
model {
    vth "Control Threshold Voltage" def=2.4 type=number tip="Control signal
```

## Virtuoso Analog Design Environment GXL User Guide

### OpenDCM

---

```
threshold voltage [V]" req=yes
    swtype "Type: Normally" def="Open" type=combo tip="Select Normally Open or
Normally Closed" val="Open,Closed" req=yes
}
advanced {
    options {
        comments
    }
}
}
```

## Verilog-A Block

You specify items that will appear on the Verilog-A[MS] tab and on the Advanced Verilog-A form in the Verilog-A block of the .gui file.

### Verilog-A {

```
sectionName {
    "sectionDescription"
    table {
        table
    }
    parameters {
        sweepVariables
    }
}

options {
    [Calibrate]
    [Verify]
}

model {
    modelParameterDefinitions
}

advanced {
    sectionName "behaviortext" [modelExtras]
    options {
        optionSpecifiers
    }
}

}
```

Folders that appear in the *Behaviors* group box and contain modeling method information

*Options* group box items

Items that appear in the *Options* group box on the Verilog-A[MS] tab

Group boxes containing advanced model options, check boxes, and data entry fields that appear on the Advanced Verilog-A form when you click *Advanced* on the Verilog-A[MS] tab

End of Verilog-A block

See also “DCM\_PARAM\_VA” on page 486 for information about associating Verilog-A code or parameter names with DCM parameters.

### Model Behaviors Group Box

You can specify the items that appear in the *Behaviors* group box on the Verilog-A[MS] tab. Each item appears as a folder that contains information such as the available modeling methods and what parameters you can sweep. You specify each folder as a section in the Verilog-A block of the .gui file as follows:

```
sectionName {
    "sectionDescription"
    table {
        table
    }
    parameters {
        sweepVariables
    }
}
```

<i>sectionName</i>	Section name. Valid Values: Any single-token string beginning with <code>section</code> and containing one or more characters (numbers, letters, or <code>_</code> ) that uniquely identify each section.
<i>sectionDescription</i>	Label for the folder that appears in the <i>Behaviors</i> group box and title text for the group box that appears to the right of the <i>Behaviors</i> group box.
<i>sweepVariables</i>	One or more sweep parameters that appear as check boxes in the <i>sectionDescription</i> group box.

For example:

```
section1 {
    "Timing model method"
    table {
        table
    }
    parameters {
        slope
        vdd
        temperature
        load_cap
    }
}
sectionMySection {
    "Input capacitance model method"
    table {
        Table
    }
    parameters {
        vdd
        temperature
        load_cap
    }
}
sectionBBFmethod {
    "Baseband Filter method 1"
    table {
        table
    }
    parameters {
```

```
        inp_freq  
    }  
}
```

### Options Group Box on Verilog-A[MS] Tab

You can specify the following keywords in the `options` section of the `Verilog-A` block of the `.gui` file to cause certain check boxes to appear in the *Options* group box on the `Verilog-A[MS]` tab:

Keyword	Description
Calibrate	Causes the <i>Calibrate</i> check box to appear.
Verify	Causes the <i>Verify</i> check box to appear.

For example:

```
options {  
    Calibrate  
    Verify  
}
```

### Model Options for Verilog-A

The items you specify in the `model` section in the `Verilog-A` block of the `.gui` file appear in the *Options* group box on the `Verilog-A[MS]` tab.

```
model {  
    modelParameterDefinitions  
}
```

where `modelParameterDefinitions` is one or more model generation options defined using the following syntax:

```
modelOption "modelOptionDescription" [modelOptionExtras]
```

`modelOption`      Name that will be used in the template as a variable that can be used in the generated model.

`modelOptionDescription`

Text that appears on the Verilog-A Model Options form next to the entry field or check box.

and *modelOptionExtras* is zero or more of the following additional specifiers:

<code>def=defaultValue</code>	Specifies a default value for the option.
<code>req= { yes   no }</code>	Specifies whether the option requires a value.  yes                    You must specify a value no                    You do not have to specify a value  Default Value: no
<code>type=type</code>	Type of value Valid Values:  sw                    Appears as a check box  combo values= <i>valueList</i>  Appears as a selection-only combo box; <i>valueList</i> is a quoted, comma-separated list of entries in the combo box  comboentry values= <i>valueList</i>  Appears as a combo box supporting both item selection and data entry; <i>valueList</i> is a quoted, comma-separated list of entries in the combo box
	int                    Appears as a data entry field; value must be an integer  num                    Appears as a data entry field; value must be a number  ent                    Appears as a data entry field; value can be any string
<code>min=minLimit</code>	(Optional) Minimum valid value for int or num type ( <code>type=int</code> or <code>type=num</code> )
<code>max=maxLimit</code>	(Optional) Maximum valid value for int or num type ( <code>type=int</code> or <code>type=num</code> )
<code>tip="optionTip"</code>	Specifies the tip text that appears in the lower left corner of the status bar of the Virtuoso Design Characterization and Modeling window when the cursor first hovers over the item on the Verilog-A Model Options form

For example:

```
model {
    tt "Clock accuracy" def=1p tip="Time accuracy of clock edge detection"
    mp "Minpulse" def=0 type=sw tip="Enable to characterize and model state
chang glitch generation. Keep output load small."
}

model {
    logtype "Natural or Decimal Log." def="Natural" type=combo
    tip="The type of logarithm used, ln(x) is natural, log(x) is decimal"
    val="Natural,Decimal" req=yes
}
```

**Note:** You can use the [DCM\\_VA\\_VALUE](#) function in test and model templates to access *modelOption* values.

## Advanced Model Options for Verilog-A

You can define the content of the [Advanced Verilog-A form](#) in the advanced section of the [Verilog-A](#) block of the .gui file as follows:

```
advanced {
    sectionName "behaviorText" modelExtras
    ...
    options {
        optionSpecifiers
    }
}
```

*sectionName*

Each *sectionName* line corresponds to a row in the table on the [Advanced Verilog-A form](#) and must match the *sectionName* you specified in the [Verilog-A](#) block.

*behaviorText*

Text that appears in the *Behavior* column as the row heading.

*modelExtras*

One or more of the following settings:

*def=defaultValue*

Specifies a default value for the option (maps to the value in the *Default* column).

*tol=verTol*

Specifies a default verification tolerance for the option (maps to the value in the *Verification tolerance* column).

*control=ctrl\_string*

Specifies the control string for the Spectre circuit simulator \$table\_model—see "Interpolating with Table Models" in the [Cadence Verilog-A Language Reference](#) (maps to the *Table control string* column).

*optionSpecifiers*      Specifies zero or more check boxes you want to see on the [Advanced Verilog-A form](#).

Valid Values (one on each line):

vdd_sample	<i>Dynamic Vdd sampling</i>
debug	<i>Add debug</i>
comments	<i>Add comments</i>

**Note:** The *Include file in header* and *Script output in header* data entry fields are always available on the form.

For example:

```
advanced {
    sectionMySection1 "Delay"           def=1n   tol=5% control=1CL
    sectionMySection2 "Output slope"    def=1n   tol=5%
    options {
        vdd_sample
        debug
        comments
    }
}
```

## Verilog-D Block

You can specify zero or more of the following keywords in the *options* section of the Verilog-D block of the .gui file to cause certain fields and check boxes to appear in the *Options* group box on the [Verilog-D tab](#):

---

Keyword	Description
Calibrate	The <i>Calibrate</i> check box appears.
Constraint	The <i>Include timing constraints</i> check box appears.

---

For example:

```
options {
    Calibrate
```

```
    Constraint  
}
```

## Defaults Block

You specify data entry fields that will appear in the *Options* group box on the *Parameters* tab of the Virtuoso Design Characterization and Modeling window in the Defaults block of the .gui file as follows:

```
Defaults {  
    defaultsString[ def=[defaultValue] ]  
    —  
}
```

where *defaultString* is one or more of the following strings (including the surrounding quotation marks):

String	Description
"Transient time step"	Possible default values for all cell types
"Transient maximum time step"	
"High voltage"	
"Low voltage"	
"Threshold voltage"	
"Stimulus interval"	Possible default values for digital cells only
"Stimulus repeat"	
"Timing constraint accuracy"	Possible default values for constraint characterization of sequential cells only
"Timing constraint delay"	
"Tristate current"	Possible default values for tristate cells only

Optionally, you can specify a default value for an item (like this: `def=defaultValue`) or an empty field (like this: `def=`).

For example:

```
Defaults {  
    "Transient time step" def=  
    "Transient maximum time step" def=2  
    "Stimulus interval" def=2  
    "High voltage" def=4  
    "Low voltage" def=5  
    "Threshold voltage" def=6  
    "Stimulus repeat"  
}
```

## Liberty Block

The program currently ignores the contents of this block.

## Sweeps Block

For model generation, you can sweep any parameter you define in the [parameters](#) block. You define defaults for the sweep type and values in the `Sweeps` block of the `.gui` file as follows:

```
Sweeps {
    guiName sweepDefinition
}
```

`guiName`                    Parameter defined in the [parameters](#) block.

`sweepDefinition`        One of the following sweep definitions:

`FROM=from TO=to BY=step`

Sweep from `from` to `to` in increments of `step`. You can express the `from`, `to`, and `step` values as numbers or percentages.

`LIST=valueList`      Sweep through the comma-separated list of values. The list must contain no spaces.

For example:

```
Sweeps {
    load_cap      FROM=50% TO=150% BY=10%
    slope         FROM=50% TO=150% BY=10%
    temperature   FROM=0     TO=100   BY=10
    vdd           FROM=90% TO=110% BY=0.1
    vsweep        LIST=-2.5,0,3,3.5,5.0
}
```

## OpenDCM Test Template

An OpenDCM test template file (`.ade`) contains enough information to generate test components (such as stimulus), measures, and analyses, and to set up simulation options for a single test for a functional cell type. OpenDCM adds the DC sources for power and ground pins automatically (unless you specify `disable_default_supplies` in the [options](#) section of the `function` block) and completes the `.ade` file by adding header, pin connection, and simulation information. You can define several tests for a single functional cell type. The program merges the processed test template with the [selected ADE state](#).

**Note:** See also “[Pin Type Definitions](#)” on page 435 for information about power and ground pin types.

OpenDCM expects to find test template files in the *targetSimulator* subdirectory of the cell-type subdirectory (see “[OpenDCM Directory Structure](#)” on page 431). For example:

Samples  
o\_myand.gui  
o\_myand  
spectre  
timing.ade

You can use any of the following OpenDCM blocks, statements, commands, and functions in a test template file (.ade):

---

<b>Block/Statement</b>	<b>Description</b>
<code>COMPONENT ...}</code>	Declares a component or measure
<code>DCM_GENERATE(...)</code> BEGIN ... DCM_END_GENERATE	Generates one or more instances of the defined component(s), uniquely named according to each pin name specified in the DCM_GENERATE function call.
<code>DCM_FOREACH idx (pinType)</code> ... DCM_END_FOREACH	Generates one line for each pin of a single <i>pinType</i> .
<code>DCM_IF(expression)</code> ... <code>DCM_ELSEIF(expression)</code> ... <code>DCM_ELSE</code> ... <code>DCM_ENDIF</code>	Conditionally generates code based on whether <i>expression</i> evaluates to true (non-zero) or false (0). You can use this construct inside or outside a DCM_GENERATE block.
<code>DCM_DEFINE macroName...</code>	Defines a substitution macro.
<code>DCM_ERROR errorMessage</code>	Defines error message text.
<code>DCM_GET_VALUE(internal)</code>	Returns the value of the specified internal variable.
<code>DCM_SKIP_TEST</code>	Suppresses generation of the .ade file.
<code>DCM_BEGIN_PERL</code> ... DCM_END_PERL	Generates custom code in OpenDCM test and model template files.
<code>DCM_BEGIN_PRIMITIVE</code> ... DCM_END_PRIMITIVE	Creates a primitive or UDP for a Verilog model.
<code>DCM_SIMULATOR</code> <code>DCM_IS_SIMULATOR</code>	Determines the simulator.
<code>DCM_WAVE</code>	Creates PWL data from digital 0's and 1's

---



Instance and net names that have a / prefix are schematic names (such as /PORT1). For each schematic name, the netlister creates a unique name that it then uses as an instance or net name in the simulator netlist. If you have any instances or nets in the netlist section of the .ade file that are not schematic-based items, you must not include the / prefix on the names when you refer to them from other DCM\_FILE sections. For example:

```
...
DCM_FILE netlist
PORT2 (RFout 0) port r=50 num=2 type=dc
PORT1 (RFin 0) port r=50 num=1 type=sine freq=dcm_frf dbm=dcm_prf \
freq2=dcm_frf2 dbm2=dcm_prf pacdbm=dcm_prf mag=1 fundname="frf" \
fundname2="frf2"
...
DCM_FILE analyses
analysis(sp fields enable) (t)
analysis(sp fields iprobe) "PORT1"
analysis(sp fields oprobe) "PORT2"
analysis(sp fields donoise) "yes"
...
```

See also [“Continuation Character for Test and Model Template Files”](#) on page 468.



The DCM\_GENERATE, DCM\_IF, and DCM\_FOREACH blocks can be nested up to a depth of 100—arbitrarily limited to 100 to prevent run-away loops caused by errors in template files.

## **DCM\_GENERATE**

You can use the DCM\_GENERATE function in test and model templates either in a single statement or as a block of statements. See the following topics for more information:

- [DCM\\_GENERATE Block](#) on page 457
- [DCM\\_GENERATE Statement](#) on page 459

### **DCM\_GENERATE Block**

DCM\_GENERATE is used in a test or model template to iterate through a set of pins of a particular type, and generate instances for any components and measures declared in the

body of the block that begins with the BEGIN keyword and ends with the DCM\_END\_GENERATE command:

```
DCM GENERATE(pinDeclarations) BEGIN
COMPONENT componentName {
    ...
}
...
DCM_END_GENERATE
```

where *pinDeclarations* is one or more pin names or *pinType* substitutions (see “[OpenDCM Pin Type Substitutions](#)” on page 477). Any pin referenced in the body of the block but not declared in *pinDeclaration* must be specified using a *pinIndex*. For example:

```
DCM GENERATE(in) BEGIN
    dcm_{in}_value = 2;           // <-- no index, iterates through pin names
    dcm_{out[1]}_value = 3;     // <-- index required
DCM_END_GENERATE
```

**Note:** If there are no pins of the declared type, then the entire block is not generated. DCM\_GENERATE does not iterate through differential pins. DCM\_IF can be used to generate conditional code for differential pins (see “[DCM\\_IF Block](#)” on page 460).

DCM\_GENERATE uses the *componentName* as the root name for the instances it generates, then appends an underscore followed by each pin name to the root name to make each name unique. DCM\_CALIBRATE and DCM\_VERIFY follow the same convention so that names in the test template are consistent with those in the model template (see “[DCM\\_CALIBRATE](#)” on page 471 and “[DCM\\_VERIFY](#)” on page 476).

The DCM\_GENERATE block can contain one or more COMPONENT blocks (see “[COMPONENT Block](#)” on page 464). DCM\_GENERATE blocks can be nested (limited to a depth of 100).

For example:

```
DCM GENERATE(in,out) BEGIN
COMPONENT meas_delay {
    ...
}
DCM_END_GENERATE
```

For each combination of pins of type *in* and *out*, a *meas\_delay* instance is created and uniquely named using the pin names. So, if pins A and B are of type *in*, and X and Y are of type *out*, then the resulting instance names become:

```
meas_delay_A_X
meas_delay_B_X
meas_delay_A_Y
meas_delay_B_Y
```

## **DCM\_GENERATE Statement**

If only a single statement requires instance generation, then the `DCM_GENERATE` function call can be used in a test or model template as a single statement without the `BEGIN` keyword and `DCM_END_GENERATE` command:

```
DCM_GENERATE(pinDeclarations) statement
```

where a `pinType` substitution in `pinDeclaration` can contain a range (for example, `out [2 : LAST]`). For example:

```
DCM_GENERATE(out[2:LAST]) real {out}_delay;
```

Here's another example:

```
DCM_GENERATE(out[2:LAST]) V({out})<+transition({out[1]})_high,{out}_delay);
```

The above statement results in the following lines being written to the model file when pins `outa`, `outb`, and `outc` are of type `out`, and `outc` is specified as the primary output pin:

```
V(outa) <+ transition(outc_high,outa_delay);  
V(outb) <+ transition(outc_high,outb_delay);
```

Here's an example where `pinDeclaration` includes two pin types:

```
DCM_GENERATE(in,out) V(N_{in}_{out}) <+ V({in}) + 2*V({out});
```

The above `DCM_GENERATE` statement results in the following four statements being written to the model file when pins `A` and `B` are of type `in`, and `X` and `Y` are of type `out`:

```
V(N_A_X) <+ V(A) + 2*V(X);  
V(N_B_X) <+ V(B) + 2*V(X);  
V(N_A_Y) <+ V(A) + 2*V(Y);  
V(N_B_Y) <+ V(B) + 2*V(Y);
```

## **DCM\_FOREACH Block**

The `DCM_FOREACH` block generates one block for each pin of a specified type:

```
DCM_FOREACH idx (pinType)  
...  
DCM_END_FOREACH
```

where

*idx* is an indexing variable used in the loop

*pinType* is a pin type substitution

The `idx` value increments for every pin of type `pinType`. Everywhere `idx` occurs in the loop, a pin of type `pinType` is substituted.

For example, the following `DCM_FOREACH` block can be used to generate one line of code for each combination of pins of type `in`:

```
DCM_FOREACH X (in)
  DCM_FOREACH Y (in)
    DCM_IF(DCM_PIN_INDEX({X}) < DCM_PIN_INDEX({Y}))
      {X}_{Y}_diff = abs(V{X}) - V({Y});
    DCM_ENDIF
  DCM_END_FOREACH
DCM_END_FOREACH
```

**Note:** `DCM_FOREACH` blocks can be nested (limited to a depth of 100) in a way that iterates a `pinType` more than once. Component name generation is not automated the way that it is in a `DCM_GENERATE` block.

## DCM\_IF Block

The `DCM_IF` block can be used inside or outside a `DCM_GENERATE` block to generate conditional lines of code in test and model templates. The following constructs are used to build the conditional block:

```
DCM_IF(expression)
DCM_ELSIF(expression)
DCM_ELSE
DCM_ENDIF
```

where `expression` is a Perl expression that evaluates to false (0) or true (non-zero), and can contain any of the following items, alone or in combination:

<b>expression Items</b>	<b>Description</b>
<code>{pinType}</code>	Returns the number of pins of type <code>pinType</code> ; anything greater than 0 evaluates to "true"
<code>{pinType[i]}</code>	Returns the value of the <i>i</i> th pin of type <code>pinType</code> ; anything greater than 0 evaluates to "true"
<code>{!pinType}</code>	Returns the number of differential pins of type <code>pinType</code> ; anything greater than 0 evaluates to "true"  <b>Note:</b> Inside a <code>DCM_GENERATE</code> block, the <code>DCM_IF({!pinType})</code> statement generates a 1 or a 0 for each instance. Outside a <code>DCM_GENERATE</code> block, the <code>DCM_IF({!pinType})</code> statement returns the total number of differential pins of type <code>pinType</code> .
<code>DCM_INV({pinType})</code>	Returns true (1) if <code>pinType</code> is an inverted pin (see " <a href="#">DCM_INV</a> " on page 481)

<b>expression Items</b>	<b>Description</b>																					
<u>DCM_IS_SIMULATOR( <i>simulatorName</i> )</u>	Returns true (1) when the selected simulator is <i>simulatorName</i> ; otherwise, returns false (0)																					
DCM_LIB_POWER	Returns true (1) when the <i>Acquire power</i> check box is marked in the <i>Options</i> group box on the <i>Liberty</i> tab; returns false (0) when the <i>Acquire power</i> check box is not marked																					
DCM_PIN_INDEX({ <i>pinType</i> })	Returns the index of <i>pinType</i> for an iteration (see “ <a href="#">Pin Index</a> ” on page 478)																					
DCM_SAME_PIN({ <i>pinType1</i> }, { <i>pinType2</i> })	Returns true (1) if <i>pinType1</i> is the same as <i>pinType2</i>																					
DCM_V_CONSTRAINTS	Returns true (1) if the <i>Include timing constraints</i> check box is marked in the <i>Options</i> group box on the <i>Verilog-D</i> tab																					
DCM_VA_DYNAMIC_VDD_SAMPLE	Returns true (1) if <i>Dynamic</i> is the selected <i>Vdd sampling</i> method on the Advanced Verilog-A form																					
Logical Operators:	Logical operators for use in a valid Perl expression:  <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 5%;">■</td> <td>==</td> <td>■ "is equal to"</td> </tr> <tr> <td>■</td> <td>&lt;</td> <td>■ "is less than"</td> </tr> <tr> <td>■</td> <td>&gt;</td> <td>■ "is greater than"</td> </tr> <tr> <td>■</td> <td>&amp;&amp;</td> <td>■ "and"</td> </tr> <tr> <td>■</td> <td>  </td> <td>■ "or"</td> </tr> <tr> <td>■</td> <td>^</td> <td>■ "xor"</td> </tr> <tr> <td>■</td> <td>!</td> <td>■ "not"</td> </tr> </table>	■	==	■ "is equal to"	■	<	■ "is less than"	■	>	■ "is greater than"	■	&&	■ "and"	■		■ "or"	■	^	■ "xor"	■	!	■ "not"
■	==	■ "is equal to"																				
■	<	■ "is less than"																				
■	>	■ "is greater than"																				
■	&&	■ "and"																				
■		■ "or"																				
■	^	■ "xor"																				
■	!	■ "not"																				

As a general rule, any Perl operator can be used. The `and`, `or`, and `xor` operators are not supported: use `&&`, `||`, and `^` instead.

Here are some examples:

<b>Example</b>	<b>Description</b>
<code>DCM_IF({reset})</code>	If one or more <code>reset</code> pins exist

Example	Description
DCM_IF({reset}>2)	If the number of <code>reset</code> pins is >2
DCM_IF({reset[2]})	If a second <code>reset</code> pin exists
DCM_IF(DCM_INV({reset}))	If <code>reset</code> is an inverted pin
DCM_IF({reset} && DCM_INV({reset[1]}))	If there is a <code>reset</code> pin and the first one is an inverted pin
DCM_IF(DCM_PIN_INDEX({in})==3)	If the index of pin <code>in</code> is equal to 3
DCM_IF(!DCM_SAME_PIN({output}, {thisOut}))	If the pin substitution for <code>output</code> is not equal to the pin substitution for <code>thisOut</code>

See also “[Checking for the Existence of Differential Pins.](#)”

### Checking for the Existence of Differential Pins

The following `DCM_IF` block can be used in test and model templates to check for the existence of differential pins, and to generate code accordingly:

```
DCM_GENERATE(in) BEGIN
    DCM_IF({!in})
        v_{in} = V({in}) - V({!in});
    DCM_ELSE
        v_{in} = V({in});
    DCM_ENDIF
DCM_END_GENERATE
```

### DCM\_DEFINE Statement

The `DCM_DEFINE` statement is used to define a substitution macro for test and model template files:

```
DCM_DEFINE macroName substitution
```

where `macroName` is the macro name used elsewhere in the file (between curly braces), and `substitution` is the text that gets substituted for `{macroName}` (or 1 if not specified). For example:

```
DCM_DEFINE RESET_CROSS @cross(V({reset[1]})-{reset[1]_thr,-1})
```

The `substitution` text can span more than one line as long as continuation characters are used in the definition (see “[Continuation Character for Test and Model Template Files](#)” on page 468).

A substitution macro can be redefined in the body of a test or model template file such that *macroName* can have a different value for each iteration of a DCM\_GENERATE or DCM\_FOREACH block. For example:

```
DCM_GENERATE(in) BEGIN
    DCM_IF(DCM_INV({in}))
        DCM_DEFINE IN_ONE 3.0
    DCM_ELSE
        DCM_DEFINE IN_ONE 0.0
    DCM_ENDIF
    ...
    COMPONENT meas_a_y {
        ...
        hi = {IN_ONE}
    ...
DCM_END_GENERATE
```



The *macroName* substitution can only be used in the file in which it is defined.

It is important to note that a DCM\_DEFINE in a DCM\_IF block is first saved for processing, but not evaluated until the second pass: A *macroName* defined in a DCM\_DEFINE in a DCM\_IF block will not have an actual value until the second pass. Therefore, the *macroName* should not appear as part of the *expression* in a subsequent DCM\_IF statement because it will not be defined. The last DCM\_IF statement in the following example fails because length(' {X} ') cannot be evaluated since X is not defined until the second pass:

```
DCM_IF(some condition)
    DCM_DEFINE X
DCM_ELSE
    DCM_DEFINE X someString
DCM_ENDIF

DCM_IF(length(' {X} ') > 5)
    output something
DCM_ENDIF
```



The single quotation marks around the {X} substitution in the length function are required for correct processing of the Perl expression.

## DCM\_SKIP\_TEST Directive

If DCM encounters the `DCM_SKIP_TEST` directive in a test template, the program does not create the `.ade` file:

```
DCM_SKIP_TEST
```

One possible application for the `DCM_SKIP_TEST` directive is to suppress generation of a test if a particular pin type does not exist. For example, consider a device that has an optional power down pin. If the pin exists, then the test for time-to-power-down should be created. However, if the pin does not exist, then there is no need to create the test:

```
DCM_IF({pd}==0)
    DCM_SKIP_TEST
DCM_ELSE
    ...
DCM_ENDIF
```

## COMPONENT Block

You specify a measure for a test using a `COMPONENT` block for each instance of the measure as follows:

```
COMPONENT instanceName {
    ...
}
```

where `instanceName` is unique for each component.

Note that

- You can parameterize anything in a `COMPONENT` block and use [OpenDCM functions](#) and [variables](#)
- `COMPONENT` blocks for OpenDCM can contain `DCM_GENERATE`, `DCM_FOREACH`, and `DCM_IF` blocks
- You can use `DCM_COMP_NAME` to pass the component name to an OCEAN measure.
- You can specify zero or more overriding parameter values for the component using a `PARAMS` subsection. For example:

```
COMPONENT load_cap {
    ...
    PARAMS {
        c = $load
    }
}
```

# Virtuoso Analog Design Environment GXL User Guide

## OpenDCM

---

The following *name=value* pairs can appear in a COMPONENT section:

Name	Value
LIBNAME	Name of library (directory) containing the netlist
CELL	Name of cell (directory) in the library (directory)
VIEW	Name of view (directory) of the cell (directory)
FILE	Name of file in the view (directory)
PATH	Full path to FILE  <b>Note:</b> PATH can reference environment variables.
COMPONENT_NAME	Component name  <b>Note:</b> For a netlist, COMPONENT_NAME is the name of the SPICE subcircuit or Verilog module. For other file types, COMPONENT_NAME is the name of the file without the suffix.
TYPE=O	Specifies the component type as an OCEAN script measure
CLASS=MEAS	Specifies the class of component as a measure
<p> <i>Important</i></p> <p>When you specify CLASS=MEAS, you must also specify an analysis type. See ANALYSIS, next.</p>	
ANALYSIS	(Required when you specify CLASS=MEAS) Analysis type, such as tran, ac, dc, op, noise, sp, xf
_ARTIST_CALC_OCN <i>expression</i> _END_ARTIST_CALC_SECTION	OCEAN measure
RESULTS <i>resultsType</i> [ <i>varList</i> ]	<u>Results type declaration</u> for measured results and the result variable(s) associated with the measure
PRESERVE_DOMAIN	PRESERVE_DOMAIN=1 causes domain values to be saved along with the measure values in the results file

## Virtuoso Analog Design Environment GXL User Guide

### OpenDCM

---

**Note:** You can specify the location of the file containing the component (measure) definition using either a lib/cell/view/file path or the full path and file name.

The following example is an instance of the `dcmMeasTim` SKILL function whose name is derived from the name of the first `clk` pin using a pin substitution:

```
COMPONENT {clk[1]}_tran {
    TYPE = O
    CLASS = MEAS
    ANALYSIS = tran
    PRESERVE_DOMAIN = 0
    PARAMS {
    }
    ARTIST_CALC_OCN
dcmMeasTim(`VT("{clk1[1]})" "rising"
            VT("{up[1]}") "falling"
            $dcm_interval DCM_PIN_LOW({p[1]}) DCM_PIN_HIGH({up[1]})
            20 80
            "n=DCM_COMP_NAME" )
    END_ARTIST_CALC_SECTION
    RESULTS SINGLE delay slope t1 t2
}
```

The following example uses the `dcmReadMeas` SKILL function to read \*.meas files from the current directory and saves the value under the name passed into the procedure:

**Note:** The \*.meas files result from a Verilog-A instance.

```
DCM_FILE measures
COMPONENT hold_{data}_high {
    TYPE = O
    CLASS = MEAS
    ANALYSIS = tran
    PRESERVE_DOMAIN = 0
    PARAMS {
    }
    ARTIST_CALC_OCN
dcmReadMeas("hold_{data}_high")
    END_ARTIST_CALC_SECTION
    RESULTS SINGLE {
        "{data} to {clk} hold high"
    }
}
```

The following example calls procedures that measure timing with and without power.

```
DCM_GENERATE(out,clk) BEGIN
DCM_FILE measures
# Clock active edge
DCM_IF(DCM_INV({clk}))
DCM_DEFINE clk_edge falling
DCM_ELSE
DCM_DEFINE clk_edge rising
DCM_ENDIF

COMPONENT {clk}_{out}_rising {
    TYPE = O
    CLASS = MEAS
    ANALYSIS = tran
```

```
PRESERVE_DOMAIN = 0
PARAMS {
}
ARTIST CALC_OCN
DCM_IF(DCM_LIB_POWER())
dcmMeasTimPwr( VT("{clk}") "{clk_edge}" VT("{out}") "rising"
IT("{vdd[1]}_source:p") VT("{vdd[1]}") VAR("dcm_load_cap") VAR("dcm_interval") 0 0
20 80 {out_list} "n=DCM_COMP_NAME" )
DCM_ELSE
dcmMeasTim( VT("{clk}") "{clk_edge}" VT("{out}") "rising" VAR("dcm_interval") 0 0
20 80 "n=DCM_COMP_NAME" )
DCM_ENDIF
END ARTIST CALC_SECTION
RESULTS SINGLE {
delay "rising delay from {clk,s} to {out}"
slope "rising slope of {out,s} (from {clk,s})"
t1 "calculated start time of {out,s} rising edge (from {clk,s})"
t2 "calculated end time of {out,s} rising edge (from {clk,s})"
end_val "final voltage on {out,s} after rising (from {clk,s})"
swipwr "switching power from {clk,s} to {out,s} rising"
}
}
...
DCM_END_GENERATE
```

## RESULTS

For measures or components that have results, the type and result variable(s) associated with the measure are declared for OpenDCM as follows:

```
RESULTS resultType [varList]
```

where *resultType* is one of the following specifiers:

- **SINGLE**—a single set of measured results for each sweep point
- **LIST**—a list of measured results and a domain for each sweep point

If there is only one result variable, then *varList* is optional. In all other cases, *varList* is a space-separated list of the result variables for the measure. Result variables in *varList* are taken to be suffixes of the measure or component instance (*componentName* declared on the COMPONENT statement), thus defining results as *componentName:resultName*. If no *varList* is specified, then the result is defined as *componentName*.

For example:

```
RESULTS SINGLE period width rise fall high low
```

Results are defined as *componentName:period*, *componentName:width*, etc.

```
RESULTS SINGLE
```

Single result is defined as *componentName*.

```
RESULTS LIST
```

List result for single result variable is defined as *componentName*.

## **DCM\_COMP\_NAME**

The DCM\_COMP\_NAME macro is used in a COMPONENT block typically to pass the component name into an OCEAN measure. The component name is substituted wherever OpenDCM encounters the DCM\_COMP\_NAME macro. For example:

```
ARTIST CALC_OCN
_dcmMeasTim( VT("{clk1[1]}") "rising"
    VT("{up[1]}") "falling"
    $dcm_interval DCM_PIN_LOW({p[1]}) DCM_PIN_HIGH({up[1]})
    20 80
    "n=DCM_COMP_NAME" )
-END_ARTIST_CALC_SECTION
```

## **Continuation Character for Test and Model Template Files**

You can use the backslash character to continue a single command that spans more than one line in a file as follows:

```
DCM_BITS(0 <DCM_PIN_COUNT({in}), 010101> \
          01111 <DCM_PIN_INDEX({in}), 11> \
          01 )
```

OpenDCM processes these lines as a single continuous command.

## **OpenDCM Model Template**

Model template files can be Verilog-A, Verilog-AMS, Verilog-D, or Liberty (.lib), depending on what the *targetSimulator* supports. You can use conditional statements to generate code based on selected pin combinations and characterization options. You can use the following OpenDCM items in a model template file:

---

<b>Item</b>	<b>Description</b>
DCM_ANALOG	See “ <a href="#">DCM ANALOG</a> ” on page 470
...	
DCM_END_ANALOG	
DCM_CALIBRATE	See “ <a href="#">DCM CALIBRATE</a> ” on page 471
DCM_COMMENT	See “ <a href="#">DCM COMMENT</a> ” on page 474
DCM_DEBUG	See “ <a href="#">DCM DEBUG</a> ” on page 474
DCM_ERROR <i>errorMessage</i>	See “ <a href="#">DCM ERROR</a> ” on page 499

---

# Virtuoso Analog Design Environment GXL User Guide

## OpenDCM

Item	Description
DCM_FOREACH <i>idx</i> ( <i>pinType</i> ) ... DCM_END_FOREACH	See “ <a href="#">DCM FOREACH Block</a> ” on page 459
DCM_GENERATE (...) ... DCM_GENERATE (...) BEGIN ... DCM_END_GENERATE	See “ <a href="#">DCM GENERATE</a> ” on page 457
DCM_IF ( <i>expression</i> ) ... DCM_ELSEIF ... DCM_ELSE ... DCM_ENDIF	See “ <a href="#">DCM IF Block</a> ” on page 460
DCM_LIB_AREA DCM_LIB_FUN	See “ <a href="#">Liberty (.lib) Value Access Functions</a> ” on page 489
DCM_LIB_CALIBRATE_DELAY DCM_LIB_CALIBRATE_HOLD DCM_LIB_CALIBRATE_INCAP DCM_LIB_CALIBRATE_LKGWR DCM_LIB_CALIBRATE_NOCPWR DCM_LIB_CALIBRATE_RECOVERY DCM_LIB_CALIBRATE_SETUP DCM_LIB_CALIBRATE_SWIPWR DCM_LIB_CALIBRATE_TRANSITION DCM_LIB_CALIBRATE_WIDTH	See “ <a href="#">Liberty (.lib) Model Calibration Functions</a> ” on page 489
DCM_MODULE	See “ <a href="#">DCM_MODULE</a> ” on page 475
DCM_NO_MODULE	See “ <a href="#">DCM_NO_MODULE</a> ” on page 475
DCM_IS_PARAM_SWEPT ( <i>param</i> ) DCM_IS_PARAM_SWEEP_FROM_TO_BY ( <i>param</i> ) DCM_IS_PARAM_SWEEP_LIST ( <i>param</i> ) DCM_PARAM_SWEEP_FROM ( <i>param</i> ) DCM_PARAM_SWEEP_TO ( <i>param</i> ) DCM_PARAM_SWEEP_BY ( <i>param</i> ) DCM_PARAM_SWEEP_LIST ( <i>param</i> )	See “ <a href="#">Sweep Value Access Functions</a> ” on page 485
DCM_PARAM_SWEPT ( <i>section</i> , <i>sweepVar</i> )	See “ <a href="#">DCM PARAM SWEPT</a> ” on page 487
DCM_PARAM_VA	See “ <a href="#">DCM PARAM VA</a> ” on page 486
DCM_SECTION	See “ <a href="#">DCM SECTION</a> ” on page 475

<b>Item</b>	<b>Description</b>
DCM_IS_SIMULATOR	See “ <a href="#">Simulator Access Functions</a> ” on page 497
DCM_SIMULATOR	
DCM_V_DEF_DELAY	See “ <a href="#">Verilog-D Value Access Functions</a> ” on page 488
DCM_V_TIMING	
DCM_VA_DEF_CAP	See “ <a href="#">Verilog-A Value Access Functions</a> ” on page 485
DCM_VA_DEF_DELAY	
DCM_VA_DEF_SLOPE	
DCM_VA_MODULES	See “ <a href="#">DCM_VA_MODULES</a> ” on page 476
DCM_VA_VALUE	See “ <a href="#">DCM_VA_VALUE</a> ” on page 501
DCM_VARS	See “ <a href="#">DCM_VARS</a> ” on page 476
DCM_VERIFY	See “ <a href="#">DCM_VERIFY</a> ” on page 476
DCM_BEGIN_PERL	See “ <a href="#">OpenDCM Perl Extensions</a> ” on page 502
...	
DCM_END_PERL	
DCM_INV({pinType})	See “ <a href="#">Pin Access Functions</a> ” on page 480
DCM_MAX_PIN_ATT(...)	
DCM_MIN_PIN_ATT(...)	
DCM_PIN_ATT(...)	
DCM_PIN_COUNT({pinType})	
DCM_PIN_HIGH({pinType})	
DCM_PIN_INDEX({pinType})	
DCM_PINS_LIST({pinType})	
DCM_PIN_LOW({pinType})	
DCM_PIN_THR({pinType})	
DCM_PINS_STRING({pinType})	
DCM_VDD	
DCM_VSS	

## **DCM\_ANALOG**

You can use the DCM\_ANALOG block as a placeholder to indicate where you want the program to insert code that would normally be inside an initial\_step or analog block. You only need to use the DCM\_END\_ANALOG statement when you add digital code to the model, to indicate where the analog block ends and the digital block begins.

**Note:** You should not add `analog begin...end` and `module...endmodule` statements to the model file.

## **DCM\_CALIBRATE**

You can use the `DCM_CALIBRATE` function in model templates to connect the measured test results with the generated model by indicating which test and results to use when creating the calibrated table as follows:

```
DCM_CALIBRATE(testName,measureName,in=pinType,out=pinType[,options])
```

where

- *testName* is the name of the test file (whose extension is `.ade`)
- *measureName* is the name of the measure—declared in the test template as a COMPONENT—whose results will be used to create the calibrated table

**Note:** This name is the same as the root *componentName* from the COMPONENT statement—before any DCM\_GENERATE action appends a pin name.

- *in=pinType* specifies the measure-from pin used for slope dependency, where *pinType* can be specified as `none` if slope dependency is not required: `in=none`
- *out=pinType* specifies the measure-to pin used for load dependency, where *pinType* can be specified as `none` if load dependency is not required: `out=none`
- *options* is a comma-separated list of zero or more of the following options:

---

<b><i>options</i></b>	<b>Description</b>
<code>domain=Vexpression</code>	Specifies the domain input to the table model for measures that have a list of results (sweep variable vs. measured result, where the sweep variable is the domain), where <i>Vexpression</i> is a valid Verilog-A (or Verilog-AMS) expression for the sweep variable and can contain a <u>pinType</u> substitution; for example:  <code>domain=V({out[1]})</code>
<code>noverify</code>	Disables creation of spec sheet entry for this set of results  <b>Note:</b> <code>noverify</code> is not valid for <code>DCM_VERIFY</code> .

---

<b>options</b>	<b>Description</b>
<code>result=resultName</code>	<p>For measures that have more than one result, indicates the result to be used</p> <p><b>Note:</b> <code>resultName</code> must match the name declared using the <u>RESULTS</u> statement in the library component definition.</p>
<code>measure2=measure2Name</code> <code>result2=result2Name</code>	<p>Specifies additional measured result "axis" (the Y axis) as input to the table model, where</p> <ul style="list-style-type: none"> <li>■ <code>measure2Name</code> is the name of the second measure</li> <li>■ <code>result2Name</code> is only necessary when there is more than one result associated with the measure, and indicates which result to use</li> </ul> <p><b>Note:</b> The <code>domain=Vexpression</code> option is used to specify the table index variable that will be used to access this axis of the measure. See the example at the end of this section.</p>
<code>abstol=abstolSpec</code>	<p>Specifies an absolute tolerance (as different from the relative percentage <i>Verification tolerance</i> specified on the Advanced Verilog-A form) using one of the following forms:</p> <ul style="list-style-type: none"> <li>■ <code>number+</code> indicates an absolute value +/- range in which the measured value must fall, <i>in addition to</i> meeting the specified relative percentage tolerance</li> <li>■ <code>number%</code> indicates a percentage tolerance to use <i>instead of</i> the specified relative percentage tolerance</li> <li>■ <code>number</code> indicates an absolute value +/- range in which the measured value must fall, <i>instead of</i> using the specified relative percentage tolerance</li> </ul>

**Note:** Keywords are not case-sensitive; assigned values are case-sensitive.

 *Important*

If the `measure2` option is specified, then `section1` and `section2` of the Verilog-A block in the `.gui` file must be modified so that only two axes can be swept (list only 2D and 3D in the table sections).

The following actions occur when the `DCM_CALIBRATE` function is called:

1. Collect sweep results into a table file.
2. Generate a reference to the calibrated table (file) in the model file as appropriate for the target simulator. For example, in a Spectre-targeted model file, the `$table_model` function is used.

For example:

```
{out[1]}_rise = DCM_CALIBRATE(timing,vco_comp,in=in,out=out[1],result=rise);
```

which results in the following line in the generated model file for Spectre circuit simulation:

```
out_rise = $table_model(V(vcontrol),my_vco_comp_rise_file,"1");
```

Consider the following excerpt from a test template:

```
DCM_GENERATE(in,out) BEGIN
    COMPONENT meas_delay {
        ...
    }
DCM_END_GENERATE

COMPONENT meas_period {
    ...
}
```

For every in-out pin combination, a `meas_delay` instance is generated with a unique name. So, for two input pins and three output pins, there are six `meas_delay` instances.

The measured data can be used in a model template as follows:

```
DCM_GENERATE(in,out) x_{in}_{out}=DCM_CALIBRATE(timing,meas_delay,in=in,out=out);
DCM_VERIFY(timing,meas_period);
```

For the same set of input and output pins, the same set of `meas_delay` instances result; thus maintaining a consistent correlation between the test and model templates. A single `meas_period` instance is verified. Its name need not be modified.

Further, consider the following template excerpt that makes use of the `measure2=measure2Name` option. The axis-access variable is specified using the `domain=Vexpression` option:

```
@(cross(V({up[2]},{up[1]}),0,ttol));
if (V({up[2]},{up[1]}) > 0) begin
    {out[1]}_up_current=DCM_CALIBRATE(up_test,up_iv_{out[1]},in={up[1]},
    out={up[2]},domain=va2,measure2=new);
    I({out[1]}) <+ {out[1]}_up_current*-1;
end
```

In the generated model file, the axis-access variable (`va2`) is specified as the second independent variable in the Spectre circuit simulator `$table_model` function:

```
@(cross(V(up,upb),0,ttol));
if (V(up,upb) > 0) begin
    out_up_current=$table_model(I(ibias)*-1,va2,"chpump_up_iv_out.vat","");
    I(out)<+ out_up_current*-1;
end
```

In the generated model-calibration plan, the second measure name (`new`) is specified in the space-separated list of variables of the `dcmCreateResultTable` SKILL function:

```
dcmCreateResultTable( "$result_dir/_open_VerilogA_gain_sweep-_open_up_test.res",
"$model_dir/chpump_up_iv_out.vat", "up_iv_out", "new dcm_ibias_value", "", "1",
"-1", $errmes);
```

## **DCM\_COMMENT**

If the *Add comments* check box is marked on the Advanced Verilog-A form, then the text that follows the `DCM_COMMENT` keyword is written to the model file as a comment (a line starting with `//`):

```
DCM_COMMENT This line is a comment.
```

The above `DCM_COMMENT` results in the following line being written to the model file when the *Add comments* check box is marked:

```
// This line is a comment.
```

**Note:** The comments *optionSpecifier* must be specified in the `options` subsection of the `advanced` section of the `Verilog-A` block of the `.gui` file (see [“Advanced Model Options for Verilog-A”](#) on page 452).

You can use the continuation character for comments that span more than one line.

## **DCM\_DEBUG**

If the *Add debug* check box is marked on the Advanced Verilog-A form, then the text that follows the `DCM_DEBUG` keyword is written to the model file:

```
DCM_DEBUG $display("This is a debug output line.");
```

This `DCM_DEBUG` results in the following line being written to the model file when the *Add debug* check box is marked:

```
$display("This is a debug output line.");
```

**Note:** The debug *optionSpecifier* must be specified in the `options` subsection of the `advanced` section of the `Verilog-A` block of the `.gui` file.

## **DCM\_MODULE**

`DCM_MODULE` is a placeholder to tell the program where to insert the module declaration. You can add additional include files before this marker.

## **DCM\_NO\_MODULE**

You can use `DCM_NO_MODULE` to suppress automatic creation of the module declaration, pin type definitions, and disciplines during model generation. You are responsible for creating the model definition manually.

Here is an example of how you might create a pin list manually by accessing and processing the same information that the program uses during automated generation:

```
DCM_NO_MODULE  
  
DCM_BEGIN_PERL  
  
my @pinList=split(' ','DCM_GET_VALUE(pin_list)');  
my $pins='';  
foreach my $p (@pinList)  
{  
    if ($p eq "{in}") {  
        sendToOutput("input in_i, in_q;\n");  
        sendToOutput("electrical in_i, in_q;\n");  
    } elsif ($p eq "{out}") {  
        sendToOutput("output out_i, out_q;\n");  
        sendToOutput("electrical out_i, out_q;\n");  
    } else {  
        $pins.=',' if (length($pins));  
        $pins.=bangToGlobal($p);  
    }  
}  
  
sendToOutput("inout $pins;\n");  
sendToOutput("electrical $pins;\n");  
  
DCM_END_PERL
```

This code might create the following lines:

```
output out_i, out_q;  
electrical out_i, out_q;  
input in_i, in_q;  
electrical in_i, in_q;  
inout vdd,vss;  
electrical vdd,vss;
```

## **DCM\_SECTION**

You can use the `DCM_SECTION` function to identify the tests associated with particular behaviors.

For example, the Verilog-A options would have two behaviors -- sectionMySec1 and sectionMySec2 (the section prefix is required). Test templates (.ade) that are associated with sectionMySec1 would include the macro "DCM\_SECTION(MySec1)," while the ones associated with sectionMySec2 would include the macro "DCM\_SECTION(MySec2)." This information is used to enable the required tests when creating sweeps for the required parameters in model behaviors.

```
DCM_SECTION( behaviorName )
```

For example:

```
DCM_SECTION(timing)
DCM_SECTION(constraint)
DCM_SECTION(incap)
```

## **DCM\_VA\_MODULES**

The DCM\_VA\_MODULES function returns the value of the `$AXL_DCM_VA_MODULES` environment variable. If it finds no value for `$AXL_DCM_VA_MODULES`, it returns the default location of Verilog-A modules.

## **DCM\_VARS**

DCM\_VARS is a placeholder to tell the program where to insert any variables it creates.

## **DCM\_VERIFY**

You can use the DCM\_VERIFY function in model templates to insert a specification for verification without generating a reference to a calibrated table in the generated model file:

```
DCM_VERIFY(testName,measureName[,options])
```

See "[DCM\\_CALIBRATE](#)" on page 471 for an explanation of these function arguments.

## **OpenDCM .txt Help File**

OpenDCM supports the use of plain text files (.txt) containing help information for a cell type. If a .txt help file exists for a cell type, the content appears when you select a functional cell type on the [Function](#) tab and choose [View – Function Help](#). OpenDCM expects to find the .txt help file in the directory that contains the .gui file of the same name (see "[OpenDCM Directory Structure](#)" on page 431). For example:

```
Samples
  o_myand.gui
```

o\_myand.txt

...

## OpenDCM Pin Type Substitutions

A *pinType* substitution is always enclosed in curly braces (`{...}`) except in `DCM_GENERATE` and `DCM_FOREACH` functions, where the curly braces are not required (see “[DCM\\_GENERATE](#)” on page 457, and “[DCM\\_FOREACH Block](#)” on page 459).

A *pinType* substitution can be any of the following:

<b><i>pinType</i></b>	<b>Description</b>
<code>pinName</code>	Simple pin name (for example, <code>in</code> , <code>out</code> , <code>reset</code> ), typically used only in <code>DCM_GENERATE</code> and <code>DCM_FOREACH</code> functions
<code>pinName[pinIndex]</code>	Indexed notation for referencing a specific pin by its index value (see “ <a href="#">Pin Index</a> ” on page 478)
<code>pinName,p</code> <code>pinName,s</code>	Forced pin ( <code>,p</code> ) or string ( <code>,s</code> ) notation (see “ <a href="#">Pin versus String Notation</a> ” on page 479)
<code>DCM_INV({pinType})</code>	Inverted pin of type <i>pinType</i>

A *pinType* substitution can be used in the following OpenDCM constructs:

<b>OpenDCM Construct</b>	<b>Examples</b>
<code>DCM_CALIBRATE(...)</code>	<code>DCM_CALIBRATE(test1,meas1,in=in,out=out, domain=V({out[1]}) )</code>
<code>DCM_FOREACH(pinType)</code>	<code>DCM_FOREACH(in)</code> <code>DCM_FOREACH(out[2:LAST])</code>
<code>DCM_GENERATE(pinType)</code>	<code>DCM_GENERATE(in)</code> <code>DCM_GENERATE(out[2:LAST])</code>
<code>DCM_IF({pinType})</code>	<code>DCM_IF({reset})</code>
<code>DCM_ELSIF({pinType})</code>	<code>DCM_ELSIF({reset})</code> <code>DCM_IF(DCM_INV({reset}))</code>
<code>DCM_INV({pinType})</code>	<code>DCM_INV({reset})</code> <code>DCM_INV({reset[1]})</code>

OpenDCM Construct	Examples
DCM_PIN_HIGH({pinType})	swing=(DCM_PIN_HIGH({out[1]})-DCM_PIN_LOW({out[1]}))/2.0
DCM_PIN_LOW({pinType})	

## Pin Index

The *pinIndex* for a *pinType* substitution can be used to indicate a particular pin in a set of pins of the same type:

*pinName*[*pinIndex*]

The *pinIndex* increments once for each iteration, and is always 1 for the primary pin (see “[Pin Type Definitions](#)” on page 435). For example:

in[1]

The *pinIndex* is must be specified if *pinName* is not used in the *pinDeclaration* of the DCM\_GENERATE function, even if there is only one pin of that type. For DCM\_GENERATE and DCM\_FOREACH functions, the *pinIndex* can include a range (for example, out[2:4], out[3:LAST]).

For example, the following DCM\_GENERATE block can be used to generate a delay measure from each input ({in}) to a single output ({out[1]}):

```
DCM_GENERATE(in) BEGIN
COMPONENT delay {
    ...
    signal1 = {in}
    signal2 = {out[1]}      #<-- pinIndex required
}
DCM_END_GENERATE
```

The following DCM\_GENERATE block can be used to generate delay measures from the primary VCO output to any secondary VCO outputs:

```
DCM_GENERATE(out[2:LAST]) {
COMPONENT delay {
    ...
    signal1 = {out[1]}      #<-- primary VCO output
    signal2 = {out}          #<-- out[2:LAST]
}
```

**Note:** The DCM\_PIN\_INDEX function can be used to access the *pinIndex* value for an iteration (see “[General Pin Information Access Functions](#)” on page 484).

## Pin versus String Notation

The format for a *pinName* depends on the context of its use. Schematic names have a forward slash added to the beginning of the name: */pinName*. For example:

Context:	Verilog	Schematic	String
<i>pinName</i>	reset	/reset	reset

The *,p* and *,s* options for a *pinType* specification are used as follows to force a representation:

*pinName, p*  
*pinName, s*

where

- *,p* forces the schematic name of a pin to be used in the test template and the Verilog name to be used in the model template (for example, */reset* and *reset*; */x<1>* and *x[1]*)
- *,s* forces the string name of a pin to be used (for example, *reset*; *x\_1\_*)

For schematic designs, signals in a test template are automatically prefixed with the */* character. This character is not legal for OCEAN measures. The *,s* option on the *pinType* substitution can be used to force a string representation, thus preventing the addition of */* to the pin name. For example:

```
COMPONENT meas_vco_ocn {
    PARAMS {
        signal = {out[1],s}
        dir = rising
        swing = (DCM_PIN_HIGH({out[1]})-DCM_PIN_LOW({out[1]}))/2.0
    }
}
```

## OpenDCM Functions

OpenDCM functions are categorized as follows:

- [Design Location Access Functions](#) on page 480
- [Pin Access Functions](#) on page 480
- [Parameter Value Access Function](#) on page 485
- [Sweep Value Access Functions](#) on page 485

- [Verilog-A Value Access Functions](#) on page 485
- [Verilog-D Value Access Functions](#) on page 488
- [Liberty \(.lib\) Value Access Functions](#) on page 489
- [Liberty \(.lib\) Model Calibration Functions](#) on page 489
- [Simulator Access Functions](#) on page 497
- [Special Functions and Statements](#) on page 497

## Design Location Access Functions

You can use the following functions to access information about design location from the [Design](#) tab.

Function	Description
DCM DESIGN LIBRARY	Returns the contents of the <i>Library Name</i> field
DCM DESIGN CELL	Returns the contents of the <i>Cell Name</i> field
DCM DESIGN VIEW	Returns the contents of the <i>View Name</i> field

## Pin Access Functions

OpenDCM provides the following functions to access pin information for use in test and model templates:

---

Function(s)	Cross-Reference
DCM_INV({pinType})	<a href="#"><u>DCM_INV</u></a> on page 481
DCM_PIN_HIGH({pinType})	<a href="#"><u>Pin Value Access Functions</u></a> on page 482
DCM_PIN_LOW({pinType})	
DCM_PIN_THR({pinType})	
DCM_VDD	
DCM_VSS	

<b>Function(s)</b>	<b>Cross-Reference</b>
DCM_PIN_ATT({pinType}, pinFieldAttr[, valueIndex])	<a href="#">Pin Attribute Access Functions</a> on page 483
DCM_MAX_PIN_ATT({pinType}, pinFieldAttr[, valueIndex])	
DCM_MIN_PIN_ATT({pinType}, pinFieldAttr[, valueIndex])	
DCM_PIN_COUNT({pinType})	<a href="#">General Pin Information Access Functions</a>
DCM_PIN_INDEX({pinType})	on page 484
DCM_PINS_LIST({pinType})	
DCM_PINS_STRING({pinType})	

## DCM\_INV

You can use DCM\_INV function in DCM\_IF and DCM\_ELSEIF expressions in test and model templates to evaluate whether a *pinType* is an inverted pin (see “[DCM\\_IF Block](#)” on page 460):

```
DCM_INV({pinType})
```

where *pinType* is a single pin name and must include an index specification unless the *pinType* is declared in a [DCM\\_GENERATE](#) function. For example:

```
DCM_IF(DCM_INV({reset[1]}))  
DCM_ELSEIF(DCM_INV({reset}))
```

The DCM\_INV function returns true (1) if *pinType* is an inverted pin, which it determines by checking for the presence of the :i *pinAttribute* in the .gui file (see “[Pin Type Definitions](#)” on page 435).

## Pin Value Access Functions

OpenDCM supports the following pin value access functions in both test and model template files:

Function	Description
DCM_PIN_HIGH({pinType})	Returns the correct parameter or expression for the high, low, or threshold level of <i>pinType</i> , which depends on user input and the netlist of the design; the expression can be one of the following:
DCM_PIN_LOW({pinType})	<ul style="list-style-type: none"><li>■ a constant</li><li>■ a parameter</li><li>■ a Verilog-A or Verilog-AMS expression</li></ul>
DCM_PIN_THR({pinType})	<b>Note:</b> See also <i>Important</i> note, below.
DCM_VDD	Returns the correct value, parameter, or expression for power ( $V_{dd}$ ) and ground ( $V_{ss}$ ) levels
DCM_VSS	<b>Note:</b> For devices with no power or ground pins, the correct global variable is returned.



### Important

If the `disable_default_supplies` option is specified in the `options` section of the function block of the `.gui` file (see “[Function Options](#)” on page 441), then using the conditional `DCM_IF` block is strongly recommended to handle cases where the design might not contain explicit power pins. Consider the following example, which assumes a custom supply pin type of `my_vdd` has been defined (see “[Pin Type Definitions](#)” on page 435). If no pins of type `my_vdd` are selected, then the default `Vdd` value is used (`DCM_VDD`):

```
DCM_IF({my_vdd})
    hi_vdd_val = DCM_PIN_HIGH({my_vdd[1] })
DCM_ELSE
    hi_vdd_val = DCM_VDD
DCM_ENDIF
```

Similarly, in a model template, the following code would be used:

```
DCM_IF({my_vdd})
    DCM_DEFINE VDDVAL V({my_vdd[1] })
DCM_ELSE
    DCM_DEFINE VDDVAL DCM_VDD
DCM_ENDIF
```

## Pin Attribute Access Functions

OpenDCM supports the following pin attribute access functions in both test and model template files:

```
DCM_PIN_ATT({pinType},pinFieldAttr[,valueIndex])
DCM_MAX_PIN_ATT({pinType},pinFieldAttr[,valueIndex])
DCM_MIN_PIN_ATT({pinType},pinFieldAttr[,valueIndex])
```

where

- *pinType* is a pin type substitution
- *pinFieldAttr* is a valid pin attribute which has been specified as *pinfield=pinFieldAttr* (a pinExtra) in the pinTypes block of the .gui file
- *valueIndex* is an optional index into a comma-separated string of values; if *valueIndex* is not specified, then the first value is assumed

The DCM\_PIN\_ATT function returns the value from the third column of the *Design pin types* table on the *Function* tab of the Virtuoso Design Characterization and Modeling window (see also “Pin Field Label” on page 438). The \_MAX\_ and \_MIN\_ functions return the maximum and minimum values of all the pins of the specified *pinType*.

For example, consider a divider with multiple outputs that have different divisors:

```
pinTypes {
    out "Output" def=t.* dir=out pinfield=d
...
}
```

The following pin attribute function accesses the value typed in the third column:

```
DCM_PIN_ATT({out},d)
```

For an example using the *valueIndex*, consider a device that has a *pinType* tap which has been defined in the .gui file as follows:

```
pinTypes {
    tap "Divider Tap" pinfield=lh
...
}
pinFields {
    "Tap Low/High" tip="Low,High count values; for example, 6,12"
}
```

The following pin attribute function accesses the second value (the high value) in the comma-separated string typed in the third column:

```
DCM_PIN_ATT({tap},lh,2)
```

Here is an example using the DCM\_MAX\_PIN\_ATT function:

```
DCM_BITS(000101010001010100<DCM_MAX_PIN_ATT({out},d),01>01)
```

## General Pin Information Access Functions

OpenDCM supports the following general pin information access functions in both test and model template files:

Function	Description
DCM_PIN_COUNT({pinType})	Returns the number of pins of type <i>pinType</i>  <b>Note:</b> Because the DCM_IF({pinType}) function also returns the number of pins of type <i>pinType</i> , the DCM_PIN_COUNT function is for use outside DCM_IF blocks (see “DCM_IF Block” on page 460).
DCM_PIN_INDEX({pinType})	Returns the index of <i>pinType</i> for an iteration (see also “Pin Index” on page 478)
DCM_PINS_LIST({pinType})	Returns a list of all pins of type <i>pinType</i> using Perl <code>qw</code> (quote words) syntax: <code>qw(pinName1 pinName2 ...)</code>  <b>Note:</b> Differential pins are not included in the list.
DCM_PINS_STRING({pinType})	Returns a list of all pins of type <i>pinType</i> as a string of names without parentheses, quotation marks, or other punctuation  <b>Note:</b> Differential pins are not included in the list.

**Note:** The DCM\_PINS\_LIST and DCM\_PINS\_STRING functions are used primarily in Perl extensions to expand into output strings or Perl `foreach` lists (see “[OpenDCM Perl Extensions](#)” on page 502). These functions are context-sensitive such that each returns the proper bit notation for test (.ade) and model (.va) files. This expansion is performed before the Perl is evaluated.

Here are some syntax examples:

```
DCM_BEGIN_PERL
$str_obe("testing pins DCM_PINS_STRING({in}).");
foreach my $pn (DCM_PINS_LIST({in})) { ...
...
DCM_END_PERL
```

## Parameter Value Access Function

You can use the `DCM_PARAM_VALUE` function to access the value of parameter on the Parameters tab as follows:

```
DCM_PARAM_VALUE( parameterName )
```

The program substitutes the value of the parameter where it finds this substitution in a test or model template file.

For example:

```
DCM_PARAM_VALUE(vdd)
```

If `vdd=3.0`, the program substitutes `3.0` where it finds this substitution in the template.

## Sweep Value Access Functions

The following sweep value access functions are available:

Function	Description
<code>DCM_IS_PARAM_SWEPT(<i>param</i>)</code>	Returns true (1) if <i>param</i> is enabled to be swept
<code>DCM_IS_PARAM_SWEEP_FROM_TO_BY(<i>param</i>)</code>	Returns true (1) if <i>param</i> is enabled to be swept using a linear sweep
<code>DCM_IS_PARAM_SWEEP_LIST(<i>param</i>)</code>	Returns true (1) if <i>param</i> is enabled to be swept through a list of values
<code>DCM_PARAM_SWEEP_FROM(<i>param</i>)</code>	Returns the <i>from</i> value of the sweep for this parameter
<code>DCM_PARAM_SWEEP_TO(<i>param</i>)</code>	Returns the <i>to</i> value of the sweep for this parameter
<code>DCM_PARAM_SWEEP_BY(<i>param</i>)</code>	Returns the step (increment) value of the sweep for this parameter
<code>DCM_PARAM_SWEEP_LIST(<i>param</i>)</code>	Returns the list of values for this parameter sweep

## Verilog-A Value Access Functions

OpenDCM provides the following functions to access information specified on the Verilog-A[MS] tab.

## See also

- [DCM\\_IF Block](#) on page 460 for information about the following Verilog-A value access functions:
  - DCM\_VA\_DYNAMIC\_VDD\_SAMPLE
  - DCM\_VAMS
- [DCM\\_PARAM\\_VA](#) on page 486 for information about associating Verilog-A code or parameters with DCM parameters
- [DCM\\_PARAM\\_SWEEP](#) on page 487 for information about including text in the model when a particular parameter is enabled

## **DCM\_PARAM\_VA**

You can use the `DCM_PARAM_VA` macro in the call to the `table` function to associate some Verilog-A code or a Verilog-A parameter name with a DCM parameter.

```
DCM_PARAM_VA( paramName, paramValue )
```

For example:

```
DCM_PARAM_VA( vdd, vdd_value )
DCM_PARAM_VA( temperature, $temperature-273.15 )
DCM_PARAM_VA( load, {OUT}_load )
DCM_PARAM_VA( slope, {IN}_slope )
```

The corresponding associations for these parameters are as follows:

```
vdd = vdd_value
temperature = $temperature-273.15
load = {OUT}_load
slope = {IN}_slope
```

So, if the template additionally contains the following lines,

```
ref_last = last_crossing(V(ref)-ref_thr,-1);
@!(cross(V(ref)-ref_thr,-1,ttol)) if (V(reset)<reset_thr) begin
    if (($abstime>dead_region) && !up_fun) begin
        if (!dn_fun) begin
ref_up_tran_delay=DCM_CALIBRATE(timing,up_ref_tran:delay,SINGLE,IN=ref,OUT=up);
```

the program writes the following to the output file:

```
ref_last = last_crossing(V(ref)-ref_thr,-1);
@!(cross(V(ref)-ref_thr,-1,ttol)) if (V(reset)<reset_thr) begin
    if (($abstime>dead_region) && !up_fun) begin
        if (!dn_fun) begin
ref_up_tran_delay=$table_model($temperature-273.15,vdd_value,"pfu_up_ref_tran_delay.vat","");
;
```

## **DCM\_PARAM\_SWEPT**

You can use the `DCM_PARAM_SWEPT` block to include text in a model when a particular parameter is enabled for sweeping for Verilog-A[MS] model calibration. See also “[DCM\\_PARAM\\_VA](#)” on page 486 for information about associating Verilog-A code or parameters with DCM parameters.

For a single line of text, you can use the following syntax:

```
DCM_PARAM_SWEPT( sectionName, parameterName )  
    Single included line of text
```

For more than one line of text, you need to use the block syntax as follows:

```
DCM_PARAM_SWEPT( sectionName, parameterName ) BEGIN  
    First included line of text  
    Second included line of text  
DCM_END_PARAM_SWEPT
```

The program includes the line or lines of text you specify in the model.

For example:

```
DCM_MODULE  
DCM_PARAM_SWEPT(section1,slope)  
    Included line because slope is swept  
DCM_PARAM_SWEPT(section1,vdd)  
    Included line because vdd is swept  
DCM_VARS  
DCM_PARAM_SWEPT(section1,temperature) BEGIN  
    Included block because temperature is swept  
    Second line of temperature block  
DCM_END_PARAM_SWEPT  
DCM_ANALOG  
DCM_PARAM_SWEPT(section1,vdd) BEGIN  
    Included block because vdd is swept  
    Second line of vdd block  
DCM_END_PARAM_SWEPT  
  
DCM_PARAM_SWEPT(section1,vdd) BEGIN  
    Even more lines to include  
DCM_END_PARAM_SWEPT  
DCM_END_ANALOG  
...
```

The output looks like this:

```
<DCM_MODULE marks the beginning of the module declaration>  
Included line because slope is swept  
Included line because vdd is swept  
  
<DCM_VARS marks insertion of variable declarations>  
Included block because temperature is swept  
Second line of temperature block  
  
<DCM_ANALOG marks the beginning of the analog block>  
Included block because vdd is swept  
Second line of vdd block
```

*Even more lines to include*

```
<end of analog block>
...
<end of module>
```

## Verilog-D Value Access Functions

OpenDCM provides the following functions to access information specified in the *Timing model method* group box on the Verilog-D tab. These functions can be used in OpenDCM test or model template files:

Function	Description
DCM_V_DEF_DELAY	Returns the delay value specified in the <i>Default</i> field
DCM_V_TIMING	Returns the selected <i>Timing model method</i> ; one of the following: <ul style="list-style-type: none"><li>■ Specify block</li><li>■ Input</li><li>■ Output</li></ul>

See also “[DCM IF Block](#)” on page 460 for information about the DCM\_V\_CONSTRAINTS Verilog-D value access function.

## Liberty (.lib) Value Access Functions

OpenDCM provides the following functions to access information specified on the *Liberty* tab. These functions can be used in OpenDCM test or model template files:

Function	Description
DCM_LIB_AREA	Returns the <i>Area</i> value.
DCM_LIB_FUN	Returns the output function, specified as a Synopsys Liberty .lib Boolean expression, that models additional functionality not currently supported in DCM. The program appends this functionality to the existing DCM-generated functionality for each output. For example, you can use the Synopsys Liberty .lib Boolean expression <code>&amp;&amp; powerdown</code> to apply the <code>powerdown</code> input pin to force all outputs to a defined state.  <b>Note:</b> The program applies the specified output function to all output pins.
DCM_LIB_POWER	Returns true (1) when the <i>Acquire power</i> check box is marked in the <i>Options</i> group box on the <i>Liberty</i> tab; returns false (0) when the <i>Acquire power</i> check box is not marked. See also “ <a href="#">DCM IF Block</a> ” on page 460.

## Liberty (.lib) Model Calibration Functions

OpenDCM provides the following functions to support calibration of Liberty (.lib) models. You can use these functions in OpenDCM model template files:

- [DCM\\_LIB\\_CALIBRATE\\_DELAY](#)
- [DCM\\_LIB\\_CALIBRATE\\_HOLD](#)
- [DCM\\_LIB\\_CALIBRATE\\_INCAP](#)
- [DCM\\_LIB\\_CALIBRATE\\_LKGPWR](#)
- [DCM\\_LIB\\_CALIBRATE\\_NOCPWR](#)
- [DCM\\_LIB\\_CALIBRATE\\_RECOVERY](#)
- [DCM\\_LIB\\_CALIBRATE\\_SETUP](#)

- [DCM\\_LIB\\_CALIBRATE\\_SWIPWR](#)
- [DCM\\_LIB\\_CALIBRATE\\_TRANSITION](#)
- [DCM\\_LIB\\_CALIBRATE\\_WIDTH](#)

### **DCM\_LIB\_CALIBRATE\_DELAY**

You can use the `DCM_LIB_CALIBRATE_DELAY` function to register the rise and/or fall delays and to write a command to the resulting template file to include the calibrated table of delays. The program also includes a calibrated table of slopes when you use the [DCM\\_LIB\\_CALIBRATE\\_TRANSITION](#) function.

```
DCM_LIB_CALIBRATE_DELAY(testName[, rise=outputRiseDelayResult][, fall=outputFallDelayResult], in=inputPin, out=outputPin[, unate=unateString])
```

*testName*                    Name of the test.

*outputRiseDelayResult*                    Name of the output rise delay result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure).

*outputFallDelayResult*                    Name of the output fall delay result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure).

*inputPin*                    [Pin type substitution](#) for the input pin.

*outputPin*                    [Pin type substitution](#) for the output pin.

*unateString*                    Valid Values:

pos                    If the *outputPin* changes as a result of the *inputPin* changing, the transition occurs in the same direction (a rising input results in a rising output; a falling input results in a falling output).

neg                    If the *outputPin* changes as a result of the *inputPin* changing, the transition occurs in the opposite direction (a rising input results in a falling output; a falling input results in a rising output).

non If the *outputPin* changes as a result of the *inputPin* changing, the transition might occur in either the same direction or the opposite direction.

Default Value: non

For example:

```
DCM_LIB_CALIBRATE_DELAY(timing, rise=a_y_rise:delay, fall=a_y_fall:delay,  
in={a[1]}, out={y[1]})
```

### **DCM\_LIB\_CALIBRATE\_HOLD**

You can use the `DCM_LIB_CALIBRATE_HOLD` function to register the hold constraint result and to write a command to the resulting template file to include the calibrated table of hold times.

```
DCM_LIB_CALIBRATE_HOLD(testName, resultName, clockInput, dataInput,  
stateString)
```

where

- *testName* is the name of the test
- *resultName* is the name of the result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)
- *clockInput* is a pin type substitution for the clock or enable input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *dataInput* is a pin type substitution for the data input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *stateString* defines the state of the *dataInput* pin as either high or low

For example:

```
DCM_LIB_CALIBRATE_HOLD(hold_data_low, hold_data_low, {clk[1]}, {d[1]}, low)
```

### **DCM\_LIB\_CALIBRATE\_INCAP**

You can use the `DCM_LIB_CALIBRATE_INCAP` function to register the calibrated input capacitance. The program replaces the function by the calibrated input capacitance.

```
DCM_LIB_CALIBRATE_INCAP(inputPin)
```

where *inputPin* is a pin type substitution for the input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477).

For example:

```
pin({in[1]}) {  
    direction : input;  
    DCM_LIB_CALIBRATE_INCAP({in[1]});  
}
```

### **DCM\_LIB\_CALIBRATE\_LKGPWR**

You can use the `DCM_LIB_CALIBRATE_LKGPWR` function to register the cell leakage power. The program replaces the function by the cell leakage power.

```
DCM_LIB_CALIBRATE_LKGPWR(testName, leakagePowerResult)
```

where

- *testName* is the name of the test
- *leakagePowerResult* is the name of the leakage power result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)

For example:

```
DCM_LIB_CALIBRATE_LKGPWR(timing, lkgpwr)
```

### **DCM\_LIB\_CALIBRATE\_NOCPWR**

You can use the `DCM_LIB_CALIBRATE_NOCPWR` function to register the no-change (internal) power and to write a command to the resulting template file to include the calibrated table of no-change power. You can use this function for cells that dissipate power when input changes do not propagate to the outputs. For example, when a flip-flop clock changes but the preset or clear is active.

```
DCM_LIB_CALIBRATE_NOCPWR(testName[, rise=inputRiseNOCPwrResult] [,  
fall=inputFallNOCPwrResult], in=inputPin)
```

where

- *testName* is the name of the test
- *inputRiseNOCPwrResult* is the name of the input rise NOC power result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)

- *inputFallNOCpwrResult* is the name of the input fall NOC power result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)
- *inputPin* is a pin type substitution for the input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)

For example:

```
DCM_LIB_CALIBRATE_NOCPWR(timing, rise=clk_rise_nocpwr, fall=clk_fall_nocpwr,  
in={clk[1]})
```

**Note:** The rise and fall edges are for the input pin.

### **DCM\_LIB\_CALIBRATE\_RECOVERY**

You can use the `DCM_LIB_CALIBRATE_RECOVERY` function to register the recovery constraint result and to write a command to the resulting template file to include the calibrated table of recovery times.

```
DCM_LIB_CALIBRATE_RECOVERY(testName, resultName, clockInput, levelInput,  
stateString)
```

where

- *testName* is the name of the test
- *resultName* is the name of the result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)
- *clockInput* is a pin type substitution for the clock or enable input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *levelInput* is a pin type substitution for the level input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *stateString* defines the state of the *levelInput* pin as either `high` or `low`

For example:

```
DCM_LIB_CALIBRATE_RECOVERY(recovery_pre, recovery_pre, {clk[1]}, {pre[1]}, low)
```

## **DCM\_LIB\_CALIBRATE\_SETUP**

You can use the `DCM_LIB_CALIBRATE_SETUP` function to register the setup constraint result and to write a command to the resulting template file to include the calibrated table of setup times.

```
DCM_LIB_CALIBRATE_SETUP(testName, resultName, clockInput, dataInput,  
stateString)
```

where

- *testName* is the name of the test
- *resultName* is the name of the result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)
- *clockInput* is a pin type substitution for the clock or enable input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *dataInput* is a pin type substitution for the data input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *stateString* defines the state of the *dataInput* pin as either high or low

For example:

```
DCM_LIB_CALIBRATE_SETUP(setup_data_low, setup_data_low, {clk[1]}, {d[1]}, low)
```

## **DCM\_LIB\_CALIBRATE\_SWIPWR**

You can use the `DCM_LIB_CALIBRATE_SWIPWR` function to register the rise and/or fall switching power values and to write a command to the resulting template file to include the calibrated table of switching power values.

```
DCM_LIB_CALIBRATE_SWIPWR(testName[, rise=outputRisePowerResult] [,  
fall=outputFallPowerResult], in=inputPin, out=outputPin[, unate=unateString])
```

where

- *testName* is the name of the test
- *outputRisePowerResult* is the name of the output rise switching power result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)
- *outputFallPowerResult* is the name of the output fall switching power result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)

- *inputPin* is a pin type substitution for the input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *outputPin* is a pin type substitution for the output pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *unateString* is one of the following:

<i>unateString</i>	Description
pos	If the <i>outputPin</i> changes as a result of the <i>inputPin</i> changing, then the transition occurs in the same direction (a rising input results in a rising output; a falling input results in a falling output)
neg	If the <i>outputPin</i> changes as a result of the <i>inputPin</i> changing, then the transition occurs in the opposite direction (a rising input results in a falling output; a falling input results in a rising output)
non	If the <i>outputPin</i> changes as a result of the <i>inputPin</i> changing, then the transition might occur in either the same direction or the opposite direction; this is the default setting

For example:

```
DCM_LIB_CALIBRATE_SWIPWR(timing, rise=a_y_rise:swipwr, fall=a_y_fall:swipwr,
in={a[1]}, out={y[1]})
```

## DCM\_LIB\_CALIBRATE\_TRANSITION

You can use the DCM\_LIB\_CALIBRATE\_TRANSITION function to register the rise and/or fall slopes. The program includes the calibrated table of slopes with the [calibrated table of delays](#).

```
DCM_LIB_CALIBRATE_TRANSITION(testName[, rise=outputRiseSlopeResult][,
fall=outputFallSlopeResult], in=inputPin, out=outputPin[, unate=unateString])
```

where

- *testName* is the name of the test
- *outputRiseSlopeResult* is the name of the output rise slope result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)

- *outputFallSlopeResult* is the name of the output fall slope result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)
- *inputPin* is a pin type substitution for the input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *outputPin* is a pin type substitution for the output pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *unateString* is one of the following:

---

<i>unateString</i>	Description
pos	If the <i>outputPin</i> changes as a result of the <i>inputPin</i> changing, then the transition occurs in the same direction (a rising input results in a rising output; a falling input results in a falling output)
neg	If the <i>outputPin</i> changes as a result of the <i>inputPin</i> changing, then the transition occurs in the opposite direction (a rising input results in a falling output; a falling input results in a rising output)
non	If the <i>outputPin</i> changes as a result of the <i>inputPin</i> changing, then the transition might occur in either the same direction or the opposite direction; this is the default setting

---

For example:

```
DCM_LIB_CALIBRATE_TRANSITION(timing, rise=a_y_rise:slope, fall=a_y_fall:slope,  
in={a[1]}, out={y[1]})
```

## **DCM\_LIB\_CALIBRATE\_WIDTH**

You can use the `DCM_LIB_CALIBRATE_WIDTH` function to register the width constraint result. The program replaces this function with the calibrated width constraint.

```
DCM_LIB_CALIBRATE_WIDTH(testName, resultName, inputPin, stateString)
```

where

- *testName* is the name of the test
- *resultName* is the name of the result (either *resultName* or *instanceName:resultName*, as written to the results file by the measure)

- *inputPin* is a pin type substitution for the input pin (see “[OpenDCM Pin Type Substitutions](#)” on page 477)
- *stateString* defines the state of the *inputPin* as either high or low

For example:

```
DCM_LIB_CALIBRATE_WIDTH(width_clock_high, width_clock_high, {clk[1]}, high)
```

## Simulator Access Functions

The DCM\_SIMULATOR macro returns the simulator being used:

```
DCM_SIMULATOR
```

You can use the DCM\_IS\_SIMULATOR function in a [DCM\\_IF](#) block to determine the selected simulator, particularly for test and model templates that support more than one simulator:

```
DCM_IF( DCM_IS_SIMULATOR( simulatorName ) )
...
DCM_ELSE
...
DCM_ENDIF
```

The program executes the first block of code if the simulator is *simulatorName*; otherwise, it executes the second block of code.

## Special Functions and Statements

OpenDCM additionally provides the following functions:

---

Function/Statement	Description
DCM_BEGIN_PRIMITIVE ... DCM_END_PRIMITIVE	Writes the enclosed block of code to \$main::design.udp, then during generation, to dcm_udp_lib/design_prim.v in the project directory (see “ <a href="#">Referencing User-Defined Primitives</a> ” on page 264)
<a href="#">DCM_ERROR</a> errorMessage	Define an error message to display
<a href="#">DCM_GET_VALUE</a> (internal)	Returns the value of the specified internal variable
DCM_LOWERCASE(string)	Converts the case of <i>string</i> to all lowercase

Function/Statement	Description
<code>DCM_SAVE_SIGNAL signalName</code>	Add DUT pins and measure ports to the saved-signal list for plotting and use in other measures
<code>DCM_UPPERCASE(string)</code>	Converts the case of <i>string</i> to all uppercase
<code>DCM_VA_VALUE(modelOption)</code>	Access Verilog-A model options
<code>PATTERN = DCM_BITS(bitStringExpr)</code>	Define a bit pattern
<code>DCM_WAVE</code>	Creates PWL name/value pairs from bit values

## DCM\_BITS

You can use the `DCM_BITS` function in a `PATTERN` statement in test and model templates to generate a string of ones and zeros for a PWL stimulus based on expressions that can include parameters or numbers of pins as follows:

```
PATTERN = DCM_BITS(bitStringExpr)
```

where *bitStringExpr* can contain any combination of the following building blocks:

Item	Description
<i>bitString</i>	String of ones and zeros
<i>repeatedBits</i>	Repeated string of the format <code>&lt;repeatFor,repeatString&gt;</code> where <ul style="list-style-type: none"> <li>■ <i>repeatFor</i> is the number of times to repeat the <i>repeatString</i>, and can be any of the following: <ul style="list-style-type: none"> <li>□ a constant</li> <li>□ an expression</li> <li>□ the result of a function call—for example, <code>DCM_VA_VALUE</code></li> </ul> </li> <li>■ <i>repeatString</i> is either a <i>bitString</i> or another <i>repeatedBits</i></li> </ul>

**Note:** Spaces in *bitStringExpr* are ignored.

For example:

```
PATTERN = DCM_BITS(0<DCM_VA_VALUE(count),01>)
```

For count=5, this PATTERN becomes:

```
00101010101
```

```
DCM_BITS(0<DCM_PIN_INDEX({in}),0<15,1>>)
```

For DCM\_PIN\_INDEX({in})=2, this PATTERN becomes:

```
00111111111111101111111111111111
```

```
PATTERN = DCM_BITS(10 <DCM_VA_VALUE(count),<DCM_PIN_COUNT({in}),0>1> 111)
```

For count=5 and DCM\_PIN\_COUNT({in})=3, this PATTERN becomes:

```
1000010001000100010001111
```

## **DCM\_ERROR**

Zero or more DCM\_ERROR functions can be used in a test or model template—typically in a conditional statement—to define an error message to display:

```
DCM_ERROR errorMessage
```

where *errorMessage* is the error message text to be displayed.

For example:

```
DCM_FOREACH x (out)
  DCM_IF(DCM_PIN_ATT({x},d) < 2)
    DCM_ERROR Divisor of pin {x} must be greater than 1
  DCM_ENDIF
DCM_END_FOREACH
```

## **DCM\_GET\_VALUE**



**Avoid using this function as it accesses the underlying data used in the current Perl implementation which might not be supported in future releases.**

The DCM\_GET\_VALUE function returns the value of an internal variable as follows:

`DCM_GET_VALUE(internal)`

For example:

<b>Function</b>	<b>Returns</b>
<code>DCM_GET_VALUE(cell_type)</code>	Name of the DCM cell functional type (a string)
<code>DCM_GET_VALUE(diff)</code>	1 if differential pin support is enabled 0 if differential pin support is disabled (see “ <a href="#">Enabling Differential Input/Output Pin Types</a> ” on page 250)
<code>DCM_GET_VALUE(create_syn)</code>	1 if the <i>Liberty (.lib) model</i> check box is marked 0 if the <i>Liberty (.lib) model</i> check box is not marked
<code>DCM_GET_VALUE(create_va)</code>	1 if the <i>Verilog-A model</i> check box is marked 0 if the <i>Verilog-A model</i> check box is not marked
<code>DCM_GET_VALUE(create_vd)</code>	1 if the <i>Verilog-D model</i> check box is marked 0 if the <i>Verilog-D model</i> check box is not marked
<code>DCM_GET_VALUE(design_name)</code>	Name of the design/cell (a string)
<code>DCM_GET_VALUE(va_cap_tol)</code>	Capacitance tolerance of the Verilog-A model (a percentage)
<code>DCM_GET_VALUE(va_tim_tol)</code>	Timing tolerance of the Verilog-A model (a percentage)
<code>DCM_GET_VALUE(va_version)</code>	Verilog-A version (a floating-point number)
<code>DCM_GET_VALUE(vd_tim_tol)</code>	Timing tolerance of the Verilog-D model (a percentage)
<code>DCM_GET_VALUE(ver_va)</code>	1 if verification of the Verilog-A model is enabled 0 if verification of the Verilog-A model is disabled

## **DCM\_SAVE\_SIGNAL**

OpenDCM automatically adds DUT pins and measure ports to the saved-signal list for plotting and use in other measures. Zero or more `DCM_SAVE_SIGNAL` statements can be used anywhere in a `.ade` file (or in a component definition in a `.lib` file) to add an intermediate or non-schematic signal to the list of saved signals from simulation:

```
DCM_SAVE_SIGNAL signalName
```

where `signalName` is a signal name specified using any of the following methods:

- A simple signal name—for example, `input`
- A signal name containing a `pinType` substitution—for example, `{in}_test` (see [“OpenDCM Pin Type Substitutions” on page 477](#))

The `.ade` file can contain zero or more `DCM_SAVE_SIGNAL` statements.

**Note:** An intermediate or non-schematic signal is one which is neither a DUT pin nor called out in any measure. One example is an internal subcircuit node.

For example:

```
DCM_SAVE_SIGNAL {in}_test1
```

## **DCM\_VA\_VALUE**

You can use the `DCM_VA_VALUE` function in test and model templates to access the value of a `modelOption` defined in the `model` section of the `Verilog-A` block of the `.gui` file.

```
DCM_VA_VALUE(modelOption)
```

where `modelOption` is the name of the variable whose value is returned. For example:

In the Verilog-A block of the `.gui` file:

```
model {  
    tt "Clock accuracy" def=1p tip="Time accuracy of clock edge detection"  
    ...  
}
```

In the model template:

```
parameter real ttol = DCM_VA_VALUE(tt);
```

**Note:** The `DCM_VA_VALUE` function returns the value specified on the Verilog-A Model Options form, or the default value (if one was specified as part of the `modelParameterDefinition`), otherwise the null string.

## DCM\_WAVE

The DCM\_WAVE function creates a set of PWL name and value pairs from digital 0 and 1 values. You provide the function with slope and digital high and low values.

```
DCM_WAVE( bits=bitList, hi=highValue, lo=lowValue, interval=stimulusInterval,
           rise_slope=riseTime, fall_slope=fallTime, delimiter=delimitChar )
```

For example:

```
DCM_DEFINE clk_pattern 00011100000000
DCM_FILE netlist
{data[1]} source ({data[1]} 0) vsource type=pwl wave=\[
DCM_WAVE(bits={data_pattern},hi=DCM_PIN_HIGH({data[1]}),\
lo=DCM_PIN_LOW({data[1]}),interval=dcm_interval,\
rise_slope=dcm_slope,fall_slope=dcm_slope,delimiter=' ')]
```

## OpenDCM Perl Extensions

In addition to the functions, substitutions, and statements outlined in “[OpenDCM Test Template](#)” on page 455 and “[OpenDCM Model Template](#)” on page 468, OpenDCM supports the use of Perl extensions for generating custom code in OpenDCM test and model template files. These Perl extensions can be used in OpenDCM templates, and OpenDCM data access functions can be used in Perl blocks. You must be familiar with Perl, DCM, and OpenDCM to use these Perl extenstions.

See the following topics for more information:

- [OpenDCM Perl Block](#) on page 502
- [Iterating through Pin Names using Perl](#) on page 503
- [Data Access Functions for Perl blocks](#) on page 504
- [Debugging Blocks of Perl Code](#) on page 507

### OpenDCM Perl Block

A block of Perl code in an OpenDCM template begins and ends with the following statements:

```
DCM_BEGIN_PERL
perlStatements
DCM_END_PERL
```

where *perlStatements* is one or more Perl statements which can contain OpenDCM functions and substitutions. The program evaluates code inside an OpenDCM Perl block as follows:

1. Process OpenDCM functions and substitutions first.
2. Evaluate Perl code using the Perl eval function.

For example, consider the following code:

```
DCM_GENERATE({in}) BEGIN
real {in} val;
DCM_PERL BEGIN
my $str = "{in}_val = V({in})+DCM_VA_VALUE(Voff);\n";
sendToOutput($str);
DCM_PERL END
DCM-END_GENERATE
```

During the first pass, the program processes OpenDCM functions and substitutions. For two pins of type in (for example, A and B), this code becomes:

```
real A_val;
DCM-BEGIN_PERL
my $str = "A_val = V(A)+1.2;\n";
sendToOutput($str);
DCM-END_PERL
real B_val;
DCM-BEGIN_PERL
my $str = "B_val = V(B)+1.2;\n";
sendToOutput($str);
DCM-END_PERL
```

During the second pass, the program executes Perl code and sends the result of my \$str to the output test (.ade) or model (.va) file using the sendToOutput Perl extension:

```
real A_val;
A_val = V(A)+1.2;
real B_val;
B_val = V(B)+1.2;
```

**Note:** The sendToOutput function guarantees that the output goes to the correct file.

## Iterating through Pin Names using Perl

The following example demonstrates how to iterate through pin names using Perl. Note that the DCM\_PINS\_LIST function returns correct Perl syntax for the list of pin names.

```
DCM-BEGIN_PERL
my $str = '';
foreach my $pn (DCM_PINS_LIST(in)) {
    if(length($str)) {
        $str .= " + V($pn)";
    } else {
        $str = V($pn);
    }
}
sendToOutput('some_signal <+ '.$str.");\n");
DCM-END_PERL
```

So, for three pins of type in (A, B, and C), DCM\_PINS\_LIST(*in*) returns `qw(A B C)`, and the following line is written to the output file:

```
some_signal <+ V(A) + V(B) + V(C);
```

## Data Access Functions for Perl blocks

You can use the following Perl extensions to access pin attributes, inverted pin information, and relative (differential) pin names.

Function	Description
<code>dcm_pin_att(\$pinName, \$attributeName, \$attributeIndex)</code>	Returns the attribute of the pin; or if no attribute is found, then it returns a null string
<code>dcm_pin_inv(\$pinName)</code>	Returns true (1) if the pin is inverted, or false (0) if the pin is not inverted
<code>dcm_pin_rel(\$pinName)</code>	Returns the relative (differential) pin name; or if no differential pin is found, then it returns a null string

**Note:** See also “[dcm\\_error](#)” on page 506.

These special Perl functions should be used instead of the standard OpenDCM `DCM_PIN_ATT` and `DCM_PIN_INV` functions whenever a Perl variable is used for a pin name (see “[Pin Attribute Access Functions](#)” on page 483 and “[DCM\\_INV](#)” on page 481). Meanwhile, the `DCM_PINS_LIST` and `DCM_PINS_STRING` pin access functions are targeted particularly for use in OpenDCM Perl blocks (see “[General Pin Information Access Functions](#)” on page 484).

For example:

```
DCM_BEGIN_PERL
my $str = '';
foreach my $pn (DCM_PINS_LIST(x1)) {
    $str .= $pn.' att b,1: '.dcm_pin_att($pn,'b',1);
    if(length(dcm_pin_att($pn,'b',2))) {
        $str .= ' att b,2: '.dcm_pin_att($pn,'b',2);
    }
    if(length(dcm_pin_att($pn,'b',3))) {
        $str .= ' att b,3 '.dcm_pin_att($pn,'b',3);
    }
    if(dcm_pin_inv($pn)) {
        $str .= " INVERTED\n";
    } else {
        $str .= "\n";
    }
}
```

```
}  
sendToOutput($str);  
DCM_END_PERL  
  
DCM_GENERATE(in) BEGIN  
DCM_BEGIN_PERL  
my $str = ',';  
foreach my $p (DCM_PINS_LIST(in)) {  
    if(length($str)) { $str .= ' && ' ; }  
    $str .= $p;  
    if(length(dcm_pin_rel($p))) { $str .= ' && '.dcm_pin_rel($p); }  
}  
$str = '{in}_signal = '.$str.";\n";  
sendToOutput($str);  
DCM_END_PERL  
DCM_END_GENERATE
```

Other files related to this example are discussed in the following sections:

- The definitions for pin types `x1` and `in` for this example, declared in the `.gui` file, are shown in [“Pin Types Defined in .gui File”](#) on page 505
- The pin assignments specified in the type table on the *Function* tab of the Virtuoso Design Characterization and Modeling window are shown in [“Pin Names and Attributes Specified in the Virtuoso Design Characterization and Modeling Window”](#) on page 505
- [Resulting Output](#) on page 506



*Important*  
Error handling inside an OpenDCM Perl block is your responsibility. See also [“Debugging Blocks of Perl Code”](#) on page 507.

### Pin Types Defined in `.gui` File

Consider the following pin type definitions for `in` and `x1` (declared in the `.gui` file) as they relate to the example shown in [“Data Access Functions for Perl blocks”](#) on page 504. Notice that pin `x1` has an inverted pin configuration:

```
pinTypes {  
    in   "Input"      def=[^v|^g].*  dir=in  
    x1   "type1"      def=w.*       dir=in tip="Type 1 Pin"      pinfield=b  
    x1:i "type1 inv" def=w.*       dir=in tip="Inverted Type pin" pinfield=b  
}
```

### Pin Names and Attributes Specified in the Virtuoso Design Characterization and Modeling Window

Consider the following pin names and attributes (specified in the pin type table on the *Function* tab of the Virtuoso Design Characterization and Modeling window and saved in the

.dcm file) as they relate to the example shown in “[Data Access Functions for Perl blocks](#)” on page 504. The first value in each `pin` statement corresponds to the *Pin Name* column entry; the second value corresponds to the *Pin Type* column entry; the third and fourth values correspond to the *Low Voltage* and *High Voltage* column entries; any additional value or string corresponds to the [optional third column of data](#) in the pin type table:

```
pin_list = a ab b bb c cb y yb vdd gnd w1 w2<3:0> w3
pin = 0 Input none none
pin = 1 !a none none
pin = 2 Input none none
pin = 3 !b none none
pin = 4 Input none none
pin = 5 !c none none
pin = 6 Output none none
pin = 7 !y none none
pin = 8 Vdd none 3.0
pin = 9 Vss 0 none
pin = 10 type1 none none 3,2,1
pin = 11 type1 none none 1,2,3
pin = 12 type1_inv none none 0
```

## Resulting Output

Combining the pin type definitions from the .gui file (see “[Pin Types Defined in .gui File](#)” on page 505) with the pin assignments made in the Virtuoso Design Characterization and Modeling window (see “[Pin Names and Attributes Specified in the Virtuoso Design Characterization and Modeling Window](#)” on page 505), the Perl code shown in “[Data Access Functions for Perl blocks](#)” on page 504 results in the following output:

```
w1 w2<3> w2<2> w2<1> w2<0> w3
w1 att b,1: 3 att b,2: 2 att b,3 1
w2<3> att b,1: 1 att b,2: 2 att b,3 3
w2<2> att b,1: 1 att b,2: 2 att b,3 3
w2<1> att b,1: 1 att b,2: 2 att b,3 3
w2<0> att b,1: 1 att b,2: 2 att b,3 3
w3 att b,1: 0 INVERTED
c_signal = a && ab && b && bb && c && cb;
b_signal = a && ab && b && bb && c && cb;
a_signal = a && ab && b && bb && c && cb;
```

## dcm\_error

This special Perl function should be used instead of the OpenDCM `DCM_ERROR` function to define an error message to display (see “[DCM\\_ERROR](#)” on page 499):

```
dcm_error("errorMessage")
```

where `errorMessage` is the error message text to be displayed.



Do not use DCM\_ERROR inside PERL blocks, as it will be evaluated before the PERL is evaluated, causing false errors.

For example:

```
DCM_BEGIN_PERL
  if(dcm_pin_att({in}, "d") < 2) {
    dcm_error("Divisor must be greater than 1.\n");
  }
...
DCM_END_PERL
```

## Debugging Blocks of Perl Code

Generally, if an error in the code causes the Perl eval function to fail, then an error message is displayed along with the block of code that caused the failure. However, it is possible to create code for which no error message is generated, but which results in undesired behavior.

For example, the my \$str statement in the following lines of code is missing the final ":".

```
DCM_BEGIN_PERL
my $str = "run=(V({in});
sendToOutput("$str\n");
DCM_END_PERL
```

The following error message could result:

```
Can't find string terminator '"' ...
In block
DCM_BEGIN_PERL
my $str = "run=(V(pinX);
sendToOutput($str);
1;
DCM_END_PERL in tmp/odcm/temp.out
Error on line 21 from file...
```

However, consider the following Perl block which contains no syntax errors:

```
DCM_BEGIN_PERL
exit;
DCM_END_PERL
```

The exit command will cause the program to exit, silently. Other Perl commands—such as reset—can cause equally puzzling behavior.



**Tip**  
OpenDCM writes the results of each intermediate processing step to a sequentially-indexed file in the `dcm/dcmDataName/odcm` directory. For debugging purposes, it can be helpful to look in the last file that OpenDCM processed before a problem occurred.

## OpenDCM Variables

OpenDCM supports the following set of pre-defined variables for use in test and model template files:

Variable	Description
<code>\$dcm_interval</code>	Stimulus interval specified on the <i>Parameters</i> tab
<code>\$dcm_maxtimestep</code>	Transient maximum time step specified on the <i>Parameters</i> tab
<code>\$dcm_temperature</code>	Temperature specified on the <i>Parameters</i> tab
<code>\$dcm_timestep</code>	Transient time step specified on the <i>Parameters</i> tab
<code>\$dcm_{vdd}_value</code>	Value of first vdd pin, if found
<code>\$dcm_{vss}_value</code>	Value of first vss pin, if found

## OpenDCM Command and Macro Reference

The following table contains an alphabetical list of OpenDCM commands, functions, blocks, statements, and macros:

Command/Macro	See:	Template Type:	
		Test	Model
<code>DCM_ANALOG</code>	<a href="#">DCM_ANALOG on page 470</a>	No	Yes
<code>DCM_BEGIN_PERL</code>	<a href="#">OpenDCM Perl Block on page 502</a>	Yes	Yes
<code>DCM_BEGIN_PRIMITIVE</code>	<a href="#">Special Functions and Statements on page 497</a>	No	Yes

**Template Type:**

<b>Command/Macro</b>	<b>See:</b>	<b>Test</b>	<b>Model</b>
DCM_BITS	<a href="#">DCM BITS on page 498</a>	Yes	Yes
DCM_CALIBRATE	<a href="#">DCM CALIBRATE on page 471</a>	No	Yes
DCM_COMMENT	<a href="#">DCM COMMENT on page 474</a>	No	Yes
DCM_COMP_NAME	<a href="#">DCM COMP NAME on page 468</a>	Yes	No
DCM_DEBUG	<a href="#">DCM DEBUG on page 474</a>	No	Yes
DCM_DEFINE	<a href="#">DCM DEFINE Statement on page 462</a>	Yes	Yes
DCM DESIGN_LIBRARY	<a href="#">Design Location Access Functions on page 480</a>	Yes	Yes
DCM DESIGN_CELL			
DCM DESIGN_VIEW			
DCM_ERROR	<a href="#">DCM ERROR on page 499</a>	Yes	Yes
DCM_FOREACH	<a href="#">DCM FOREACH Block on page 459</a>	Yes	Yes
DCM_GENERATE	<a href="#">DCM GENERATE on page 457</a>	Yes	Yes
DCM_GET_VALUE	<a href="#">DCM GET VALUE on page 500</a>	Yes	Yes
DCM_IF	<a href="#">DCM IF Block on page 460</a>	Yes	Yes
DCM_INV	<a href="#">DCM INV on page 481</a>	Yes	Yes
DCM_IS_PARAM_SWEEP	<a href="#">Sweep Value Access Functions on page 485</a>	Yes	Yes
DCM_IS_PARAM_SWEEP_FROM_TO_BY			
DCM_IS_PARAM_SWEEP_LIST			
DCM_IS_SIMULATOR	<a href="#">Simulator Access Functions on page 497</a>	Yes	Yes
DCM_LIB_AREA	<a href="#">Liberty (.lib) Value Access Functions on page 489</a>	No	Yes
DCM_LIB_CONSTRAINTS			
DCM_LIB_FUN			

**Template Type:**

<b>Command/Macro</b>	<b>See:</b>	<b>Test</b>	<b>Model</b>
DCM_LIB_CALIBRATE_DELAY	<a href="#">Liberty (.lib) Model Calibration Functions</a> on page 489	No	Yes
DCM_LIB_CALIBRATE_HOLD			
DCM_LIB_CALIBRATE_INCAP			
DCM_LIB_CALIBRATE_LKGPWR			
DCM_LIB_CALIBRATE_NOCPWR			
DCM_LIB_CALIBRATE_RECOVERY			
DCM_LIB_CALIBRATE_SETUP			
DCM_LIB_CALIBRATE_SWIPWR			
DCM_LIB_CALIBRATE_TRANSITION			
DCM_LIB_CALIBRATE_WIDTH			
DCM_LOWERCASE	<a href="#">Special Functions and Statements</a> on page 497	Yes	Yes
DCM_MAX_PIN_ATT	<a href="#">Pin Access Functions</a> on page 480	Yes	Yes
DCM_MIN_PIN_ATT			
DCM_MODULE	<a href="#">DCM_MODULE</a> on page 475	No	Yes
DCM_PARAM_SWEEP_FROM	<a href="#">Sweep Value Access Functions</a> on page 485	Yes	Yes
DCM_PARAM_SWEEP_TO			
DCM_PARAM_SWEEP_BY			
DCM_PARAM_SWEEP_LIST			
DCM_PARAM_SWEPT	<a href="#">DCM_PARAM_SWEPT</a> on page 487	No	Yes
DCM_PARAM_VA	<a href="#">DCM_PARAM_VA</a> on page 486	Yes	Yes
DCM_PARAM_VALUE	<a href="#">Parameter Value Access Function</a> on page 485	Yes	Yes
DCM_PIN_ATT	<a href="#">Pin Access Functions</a> on page 480	Yes	Yes
DCM_PIN_COUNT			
DCM_PIN_HIGH			
DCM_PIN_INDEX			
DCM_PIN_LOW			
DCM_PIN_THR			
DCM_PINS_LIST			
DCM_PINS_STRING			
DCM_SAVE_SIGNAL	<a href="#">DCM_SAVE_SIGNAL</a> on page 501	Yes	No
DCM_SECTION	<a href="#">DCM_SECTION</a> on page 475	No	Yes

Template Type:			
Command/Macro	See:	Test	Model
DCM_SIMULATOR	<a href="#">Simulator Access Functions</a> on page 497	Yes	Yes
DCM_SKIP_TEST	<a href="#">DCM_SKIP_TEST Directive</a> on page 464	Yes	No
DCM_UPPERCASE	<a href="#">Special Functions and Statements</a> on page 497	Yes	Yes
DCM_V_CONSTRAINTS	<a href="#">Verilog-D Value Access Functions</a> on page 488	Yes	Yes
DCM_V_DEF_DELAY			
DCM_V_TIMING			
DCM_VA_DEF_CAP	<a href="#">Verilog-A Value Access Functions</a> on page 485	Yes	Yes
DCM_VA_DEF_DELAY			
DCM_VA_DEF_SLOPE			
DCM_VA_DYNAMIC_VDD_SAMPLE			
DCM_VA_MODULES	<a href="#">DCM_VA_MODULES</a> on page 476	No	Yes
DCM_VA_VALUE	<a href="#">DCM_VA_VALUE</a> on page 501	Yes	Yes
DCM_VAMS	<a href="#">DCM_IF Block</a> on page 460	Yes	Yes
DCM_VARS	<a href="#">DCM_VARS</a> on page 476	No	Yes
DCM_VDD	<a href="#">Pin Value Access Functions</a> on page 482	Yes	Yes
DCM_VERIFY	<a href="#">DCM_VERIFY</a> on page 476	No	Yes
DCM_VSS	<a href="#">Pin Value Access Functions</a> on page 482	Yes	Yes
DCM_WAVE	<a href="#">DCM_WAVE</a> on page 502	Yes	No

## Creating Templates That Support Differential Pins

The following example is used to demonstrate how to create a set of OpenDCM templates (.gui, .ade, .va) that support a reset pin that can be either inverted (low true) or noninverted (high true) and that may or may not have a differential pin.

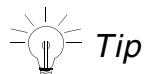
See the following topics for more information:

- [The .gui File on page 512](#)
- [The .ade File on page 512](#)
- [The .va File on page 513](#)

## The .gui File

The `pinTypes` and `check` sections of the `function` block in the `.gui` file specify that there can be zero or one `reset` pin, and that the `reset` pin can be either inverted (low true) or noninverted (high true). The `options` section has the `diff` option enabled so that there can be a differential pin.

```
function {...
    pinTypes {...}
        reset:i "Low Reset"      def=(r.*|c.*).* (n|z)    dir=in
        reset    "High Reset"    def=(r.*|c.*)
    ...
    check {...}
        reset:* < 2 "Must have a maximum of 1 Reset input"
    ...
    options {
        diff
    }
}
```



See “[function Block](#)” on page 434 for more information.

## The .ade File

The `.ade` file contains a `DCM_GENERATE` block to iterate through each pin of type `reset` to generate the drive source for each `reset` pin (see “[DCM\\_GENERATE Block](#)” on page 457). Note that `DCM_GENERATE` does not iterate through differential pins.

```
DCM_GENERATE(reset) BEGIN
COMPONENT vpulse_{reset} {
    ...
    PARAMS {
        DCM_IF(DCM_INV({reset}))
        val0 = DCM_PIN_HIGH({reset})
        val1 = DCM_PIN_LOW({reset})
        DCM_ELSE
        val0 = DCM_PIN_LOW({reset})
        val1 = DCM_PIN_HIGH({reset})
        DCM_ENDIF
    ...
}
```

```
PINS plus>{reset} minus>ground  
}
```

At this point in the DCM\_GENERATE block, a DCM\_IF statement is used to check for the presence of a differential pin. If a differential pin exists, then a component instance is generated, using the name of the differential pin to create a unique name:

```
DCM_IF({!reset})  
COMPONENT vpulse_{!reset} {  
    ...  
    PARAMS {  
        DCM_IF(DCM_INV({reset}))  
        val0 = DCM_PIN_LOW({reset})  
        val1 = DCM_PIN_HIGH({reset})  
    DCM_ELSE  
        val0 = DCM_PIN_HIGH({reset})  
        val1 = DCM_PIN_LOW({reset})  
    DCM_ENDIF  
    ...  
}  
PINS plus>{!reset} minus>ground  
}  
DCM_ENDIF  
DCM_END_GENERATE
```

Differential pins do not have separate attributes; they derive their attributes from the master pin. For this reason, the `{reset}` pin type substitution (and *not* the `{!reset}` pin type substitution) is used in the test for an inverted pin and in the pin value access functions in *both* PARAMS blocks (see “[DCM INV](#)” on page 481 and “[Pin Value Access Functions](#)” on page 482).

## The .va File

The .va file accomplishes two tasks:

- Creates a threshold variable if the `reset` pin is not differential (see “[Creating the Threshold Variable](#)” on page 513)
- Creates expressions that detect when `reset` changes, either from true to false or from false to true (see “[Detecting Crossing Points](#)” on page 514)

### Creating the Threshold Variable

If the `reset` pin is not differential, then the Verilog-A[MS] code must detect when it crosses a threshold. This task can be accomplished using the following conditional blocks in the .va file. This first block comes after the DCM\_VARS statement to create a threshold variable declaration if the `reset` pin is not differential:

```
DCM_IF({reset} && !({!reset}))  
real_{reset}_thr;  
DCM_ENDIF
```

This second block comes after the DCM\_ANALOG statement to set the value of the threshold variable to the correct value:

```
DCM_IF({reset} && !({!reset}))  
{reset}_thr = DCM_PIN_THR({reset});  
DCM_ENDIF
```

## Detecting Crossing Points

The following combination of Perl and OpenDCM statements can be used to generate the code that detects crossing points. OpenDCM functions and substitutions within a block of Perl code are processed *before* the Perl statements are evaluated (see “[OpenDCM Perl Block](#)” on page 502).

The DCM\_IF({!reset[1]}) block checks for the presence of a differential pin for {reset[1]}. If the differential pin exists, then the appropriate \$str Perl statement is evaluated *after* determining whether {reset[1]} is an inverted pin. If the differential pin does *not* exist, then the test for pin inversion is performed as part of the evaluation of the Perl statements, using the Perl if statement and the OpenDCM dcm\_pin\_inv function (see “[Data Access Functions for Perl blocks](#)” on page 504).

```
DCM_IF({reset[1]})  
DCM_COMMENT set run to 1 if {reset[1]} is false  
DCM_BEGIN_PERL  
my $str = "\trun=";  
DCM_IF({!reset[1]})  
    DCM_IF(DCM_INV({reset[1]}))  
        $str .= "(V({reset}) > V({!reset}))";  
    DCM_ELSE  
        $str .= "(V({!reset}) > V({reset}))";  
    DCM_ENDIF  
DCM_ELSE  
    $str .= "(V({reset})";  
    if(dcm_pin_inv('{reset[1]}')) {  
        $str .= '>';  
    } else {  
        $str .= '<';  
    }  
    $str .= '{reset[1]}_thr)';  
DCM_ENDIF  
$str .= ";"\n;  
sendToOutput($str);  
DCM_END_PERL  
DCM_ELSE  
DCM_COMMENT no reset pin  
...
```

---

## Environment Variables

---

This appendix describes public environment variables that control characteristics of the Analog Design Environment GXL (ADE GXL). You can customize the operation and behavior of ADE GXL features by changing the values of particular environment variables.

See

- [ADE GXL Environment Variables](#) on page 516
- [Design Characterization and Modeling](#) on page 519

Also see:

- “[Environment Variables](#)” in the [Virtuoso Analog Design Environment XL User Guide](#)

## ADE GXL Environment Variables

You can set the following environment variables in your `.cdsenv` or `.cdsinit` files to customize the settings for simulations or results:

- sensitivityPlotContinuousLine
- sensitivityThumbnailMaxPointsForLinePlot
- digitsToShowForYieldInPercentage
- sortVariablesOpt
- stopManualTuningOnSessionExit

### **sensitivityPlotContinuousLine**

Specifies if a line plot should be plotted for the results of sensitivity analysis.

In `.cdsenv`:

```
adexl.plotting sensitivityPlotContinuousLine boolean t
```

In `.cdsinit` or the CIW:

```
envSetVal("adexl.plotting" "sensitivityPlotContinuousLine" 'boolean  
t)
```

Valid Values:

t	This is the default value.
nil	

### **sensitivityThumbnailMaxPointsForLinePlot**

Specifies the minimum number of points for which line plots should be plotted in the thumbnails in the Sensitivity Analysis results window. When the number of points is more than the value of this variable, the thumbnail plots show scatter plots.

In `.cdsenv`:

```
adexl.plotting sensitivityThumbnailMaxPointsForLinePlot int t
```

In `.cdsinit` or the CIW:

```
envSetVal("adexl.plotting"  
         "sensitivityThumbnailMaxPointsForLinePlot" 'int t)
```

Valid Values:

A positive integer value

Default Value: 10

## **digitsToShowForYieldInPercentage**

Specifies the number of digits to be displayed for values in the *Yield In Percentage* column on the Results tab for High Yield Estimation run.

In .cdsenv:

```
adexl.gui digitsToShowForYieldInPercentage int 6
```

In .cdsinit or the CIW:

```
envSetVal("adexl.gui" "digitsToShowForYieldInPercentage" 'int t)
```

Valid Values:

A positive integer value

Default Value: 10

## **sortVariablesOpt**

Specifies if the variables and parameters should be sorted before generating random samples for an optimization run. By default, the variables are not sorted before the run is started. However, you can sort them by setting this variable to t so as to ensure that the result of different optimization runs is same irrespective of the order of variables and parameters.

In .cdsenv:

```
adexl.algorithm sortVariablesOpt boolean nil
```

In .cdsinit or the CIW:

```
envSetVal("adexl.algorithm" "sortVariablesOpt" 'boolean t)
```

Valid Values:

t	Sorts the variables and parameters before generating random samples.
nil	Does not sort the variables and parameters before generating random samples. This is the default value.

## **stopManualTuningOnSessionExit**

Specifies if the Manual Tuning run mode should be stopped when ADE XL GUI is closed while that run is in progress.

In .cdsenv:

```
adexl.simulation stopManualTuningOnSessionExit boolean nil
```

In .cdsinit or the CIW:

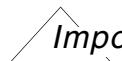
```
envSetVal("adexl.simulation" "stopManualTuningOnSessionExit"  
'boolean t)
```

Valid Values:

t	Stops the currently running Manual Tuning run when the ADE XL GUI is closed. In the next session, the Run Simulation button is green in color and you can either start a new Manual Tuning run or submit new points in the previous Manual Tuning run.
nil	Does not stop the currently running Manual Tuning run when the ADE XL GUI is closed. When you open a new ADE XL session, the Run Simulation button is yellow in color and you can continue with the same Manual Tuning run that was running earlier.  This is the default value.

## Design Characterization and Modeling

You must set the following environment variables before starting your Cadence software in order to apply these settings to your Design Characterization and Modeling (DCM) session.



You cannot set these variables in your `.cdsenv` or `.cdsinit` files.

---

Environment Variable	Description
<code>AXL_DCM_GUI</code>	Adds one or more cell type directories to the default DCM list (from <code>yourInstallDir/tools/dfII/etc/dcm/cellTypes</code> ). Separate directory paths using a colon (:). See also <a href="#">Appendix A, “OpenDCM”</a> .
<code>AXL_DCM_GUI_DEF</code>	Defines the cell type directories to use; the program does not use directories from the default DCM list (in <code>yourInstallDir/tools/dfII/etc/dcm/cellTypes</code> ). See also <a href="#">Appendix A, “OpenDCM”</a> .
<code>HOME</code>	Defines the home location for saving/loading user-specific information such as the list of recent files.

---

When you specify one or more directories containing OpenDCM cell types using the `AXL_DCM_GUI` or `AXL_DCM_GUI_DEF` environment variable, each directory appears as a design function category in the *Function* list box on the *Function* tab of the Virtuoso Design Characterization and Modeling window. For example, if you set `AXL_DCM_GUI` as follows:

```
setenv AXL_DCM_GUI $Proj1/samples/dcm/odcm/Samples:/home/myOdcn:/home/myOdcn/myC1
```

## Virtuoso Analog Design Environment GXL User Guide

### Environment Variables

---

the following design function categories appear in the *Function* list box on the *Function* tab the next time you run DCM:

*Samples*  
*myOdcm*  
*myC1*

You can use the following environment variables to apply various settings for special cases or to troubleshoot problems.

---

Environment Variable	Description
AXL_DCM_ABS_PATHS	<p>When set, specifies that you want the program to create absolute paths in generated SKILL files.</p> <p>Default Behavior: The program creates paths relative to the current directory in generated SKILL files so that you can move data between sites.</p> <p>Example:</p> <pre>setenv AXL_DCM_ABS_PATHS</pre>
AXL_DCM_KEEP_SDB	<p>When set, specifies that you want the program not to delete the existing .sdb file when you select <i>Generate</i>.</p> <p>Example:</p> <pre>setenv AXL_DCM_KEEP_SDB</pre>
AXL_DCM_MEAS_PREAMBLE	<p>Defines the name of a SKILL/OCEAN file that you want the program to include before the SKILL measurement code.</p> <p><b>Note:</b> You can use this setting to generate debug information or to define variables before the program runs the measures.</p> <p>Example:</p> <pre>setenv AXL_DCM_MEAS_PREAMBLE ~/myDCMMeasPreamble.il</pre>
AXL_DCM_NO_CONFIG	<p>Forces simulation of the design to use the schematic directly rather than using a config view where you can define the view and stop lists.</p> <p>Example:</p> <pre>setenv AXL_DCM_NO_CONFIG</pre>

## Virtuoso Analog Design Environment GXL User Guide

### Environment Variables

---

Environment Variable	Description
AXL_DCM_NO_SHUFFLE	Disables form-size and button-position adjustments that the software makes to use the Cadence style (“skin”).  Example: <pre>setenv AXL_DCM_NO_SHUFFLE</pre>
AXL_DCM_NOCMD	Disables communication between DCM and the parent application (the one that calls DCM, such as <code>icms</code> ).  <b>Note:</b> DCM does not communicate with the parent application unless you run the DCM binary ( <code>dcmPlus</code> ) with the <code>-adexL</code> argument.  Example: <pre>setenv AXL_DCM_NOCMD</pre>
AXL_DCM_NOGETSIM	When set, DCM does not request the list of available simulators from the parent application (the one that calls DCM, such as <code>icms</code> ).  <b>Note:</b> DCM does not communicate with the parent application unless you run the DCM binary ( <code>dcmPlus</code> ) with the <code>-adexL</code> argument.  Example: <pre>setenv AXL_DCM_NOGETSIM</pre>
AXL_DCM_OLD_STYLE	When set, disables the Cadence environment style (“skin”).  <b>Note:</b> You can set this environment variable if you suspect that the Cadence skin might be causing a problem on a specific machine or in a specific environment.  Example: <pre>setenv AXL_DCM_OLD_STYLES</pre>
AXL_DCM_POST_SIM_DELAY	Specifies a delay (number of seconds) after the software finishes all simulations and before it calls the next procedure (from characterization to calibration and verification to comparison).  Example: <pre>setenv AXL_DCM_POST_SIM_DELAY 2</pre>

## Virtuoso Analog Design Environment GXL User Guide

### Environment Variables

---

---

#### Environment Variable    Description

---

AXL\_DCM\_PRE\_SIM\_DELAY

Specifies a delay (number of seconds) before the program starts each simulation.

**Note:** You can use this setting if you have race conditions when your simulations start.

Example:

```
setenv AXL_DCM_PRE_SIM_DELAY 5
```

AXL\_DCM\_VA\_MODULES    Specifies the location of Verilog-A measurement modules.

Default Value:

```
yourInstallDir/tools/dfII/etc/dcm/vaModules
```

Example:

```
setenv AXL_DCM_VA_MODULES ~/myModules
```

AXL\_DCMHOME    Defines DCM's home directory where it locates cell types and other information.

**Note:** DCM gets its home directory automatically when you run it from the Virtuoso® Design Environment. If you run DCM outside the Design Environment (using the `dcmPlus` command in a terminal window), you can use this setting to define the DCM home directory.

Example:

```
setenv AXL_DCMHOME ~/myInstall/tools/dfII
```

---

# C

---

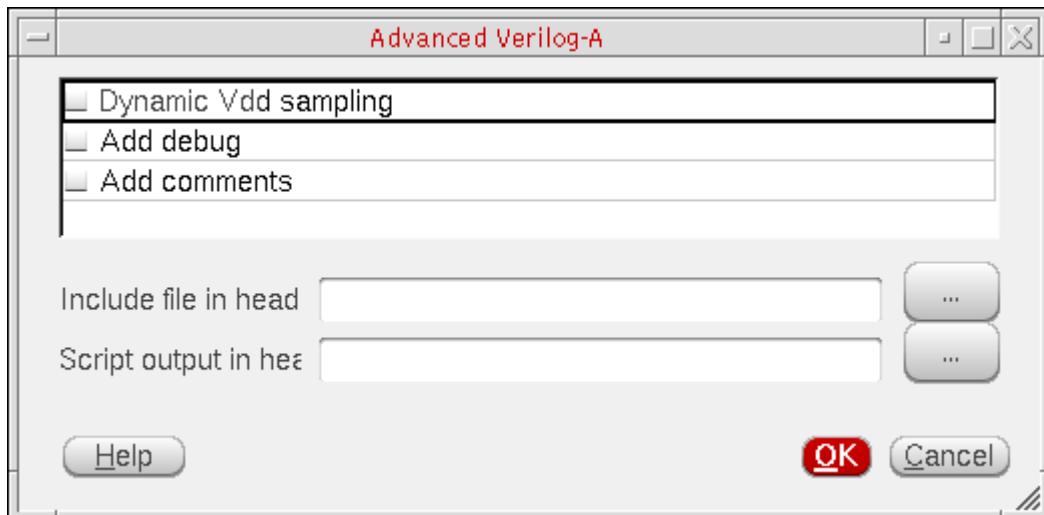
## Form Descriptions

---

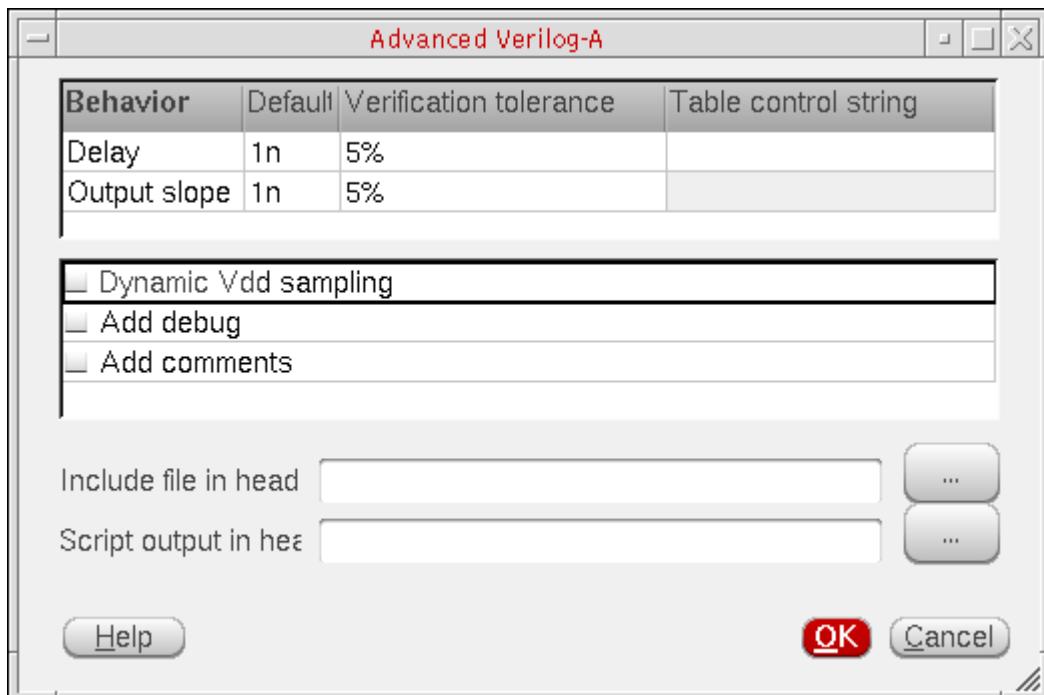
- [Advanced Verilog-A](#) on page 524
- [Choose Design](#) on page 525
- [Choose a Directory](#) on page 526
- [Design Pin Attributes](#) on page 527
- [Function Filter](#) on page 528
- [Liberty \(.lib\) Library Generation](#) on page 529
- [Model Destination](#) on page 530
- [Run Status](#) on page 532
- [Start](#) on page 533
- [Sweep Setup](#) on page 534

## Advanced Verilog-A

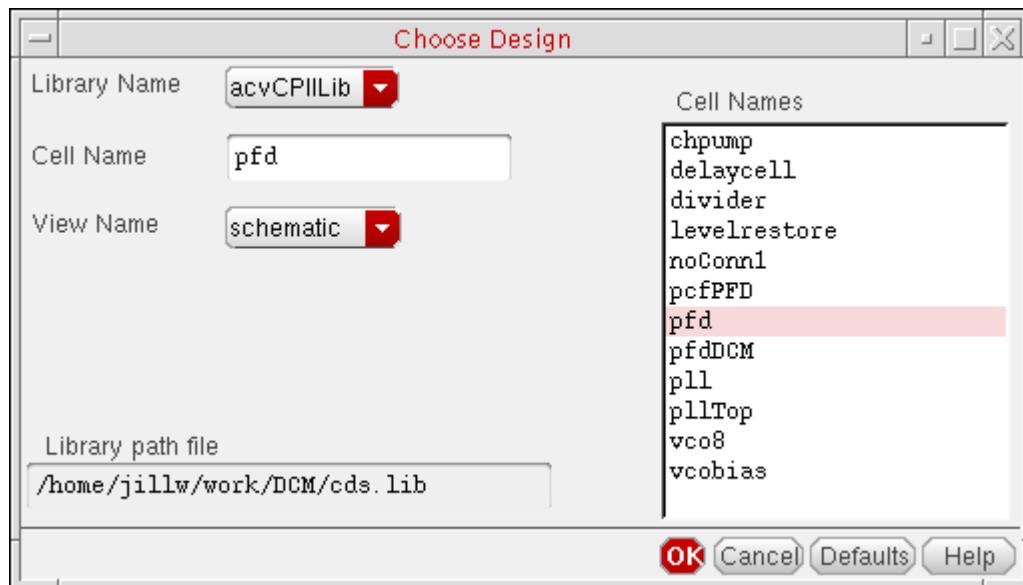
for Top-Down modeling



for Bottom-Up modeling



## Choose Design



*Library Name* is a drop-down list from which you can select a design library.

*Cell Name* is a scrolling list of cell names from which you can select a design cell.

*Cell Name* is a field in which the cell you select from the *Cell Names* scrolling list appears. Alternatively, you can type a valid cell name in this field.

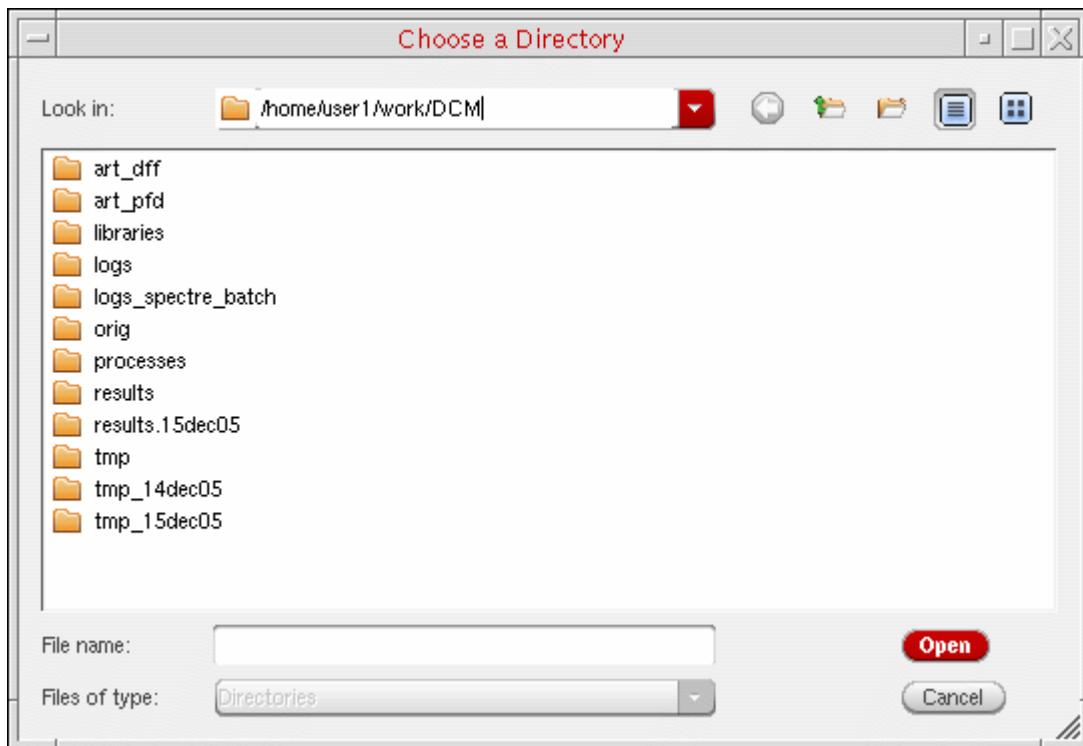
*View Name* is a drop-down list from which you can select a design view.

*OK* puts the information you selected in the fields of the *Design* tab and closes the form.

*Cancel* closes the form without performing any task.

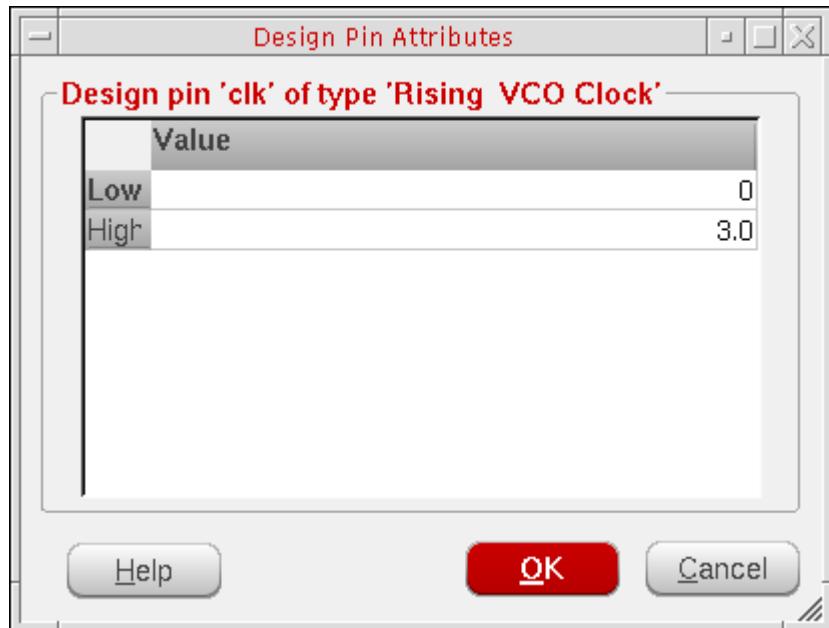
*Help* opens the Online Help page for this form.

## Choose a Directory

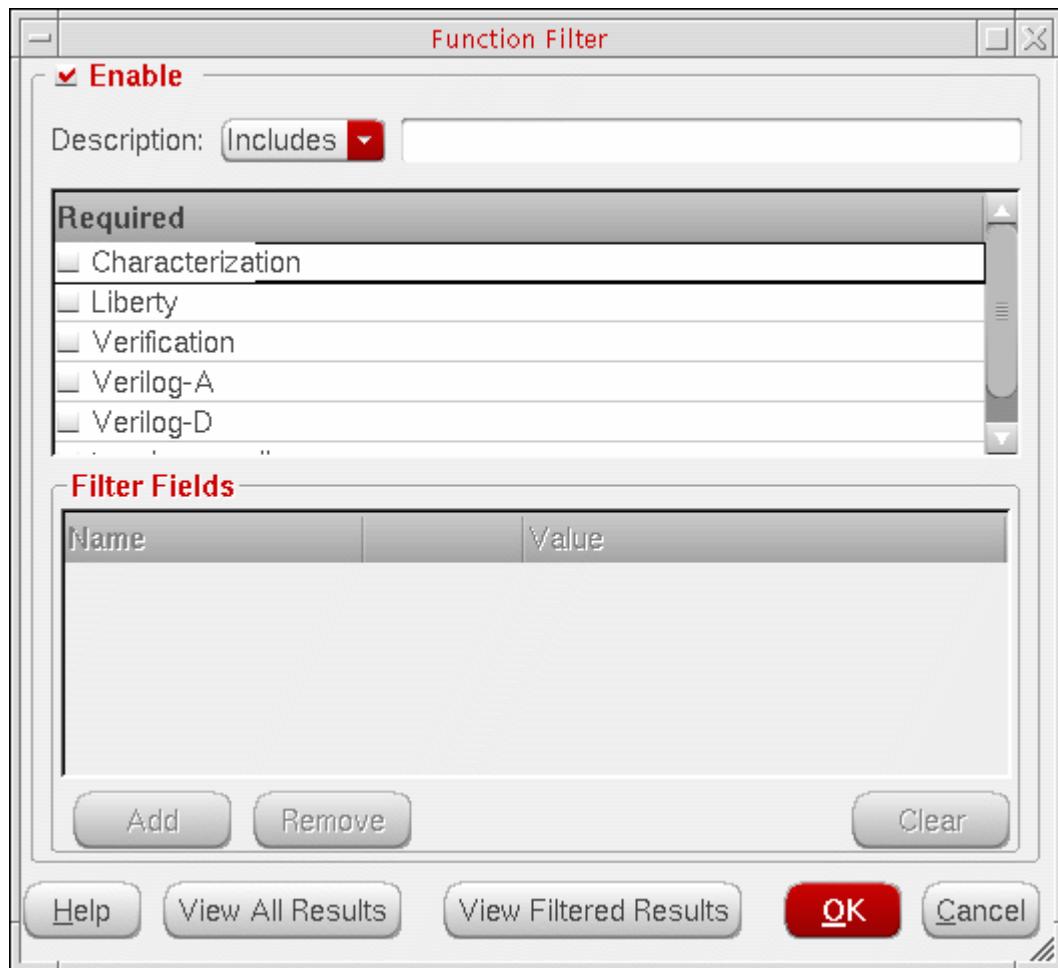


1. Use the navigation tools on the form to select an ADE state directory name.
2. Click *Open*.

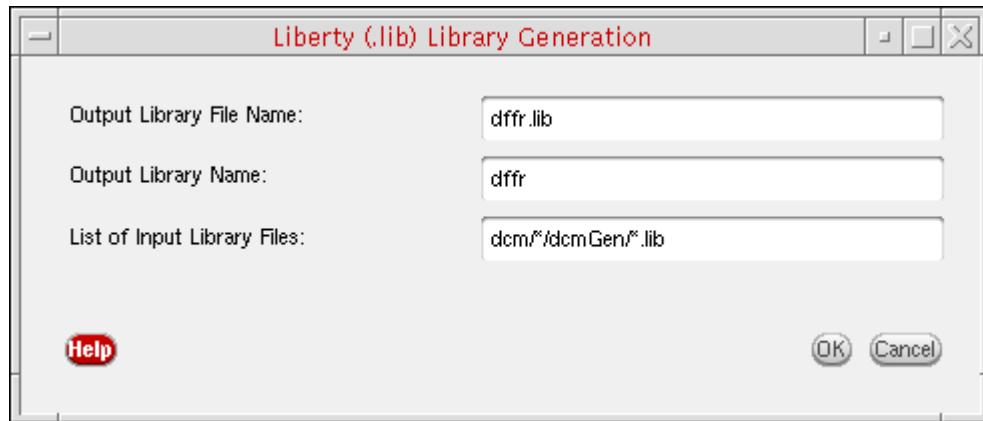
## Design Pin Attributes



## Function Filter

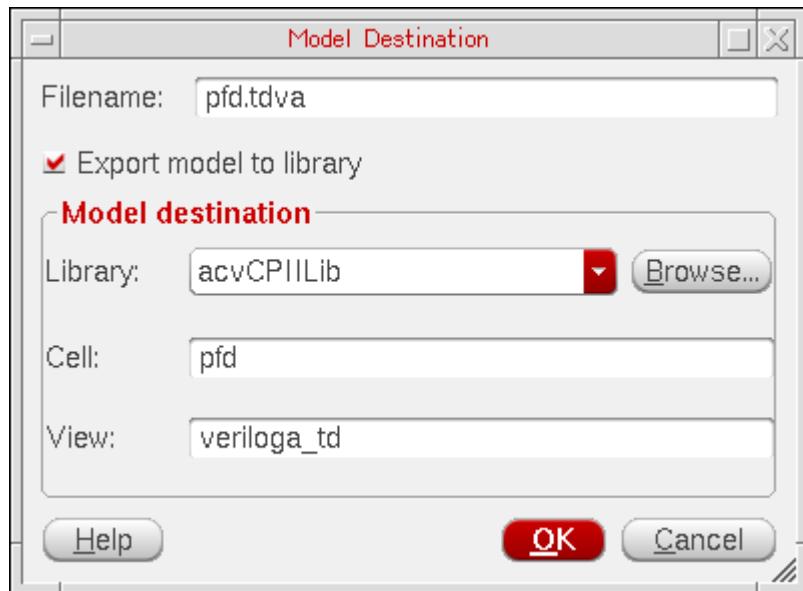


## **Liberty (.lib) Library Generation**



## Model Destination

To specify a model destination, do the following:



1. (Optional) In the *Filename* field, type a name for the behavioral model file.

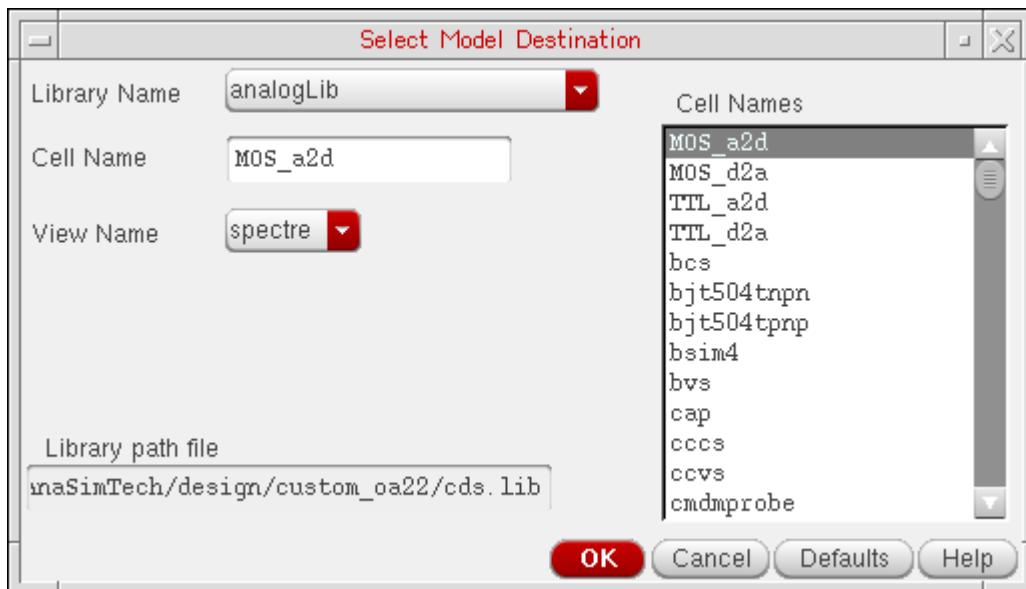
The program provides a default name in this field as follows:

- cellName.tdva* for top-down Verilog-A[MS]
- cellName.va* for bottom-up Verilog-A[MS]
- cellName.v* for bottom-up Verilog-D



(Optional) If you remove the mark from the *Export model to library* check box, the *Model destination* group box becomes inactive and you cannot specify a library destination for the generated model. The program does not copy the generated model to a library location.

- 2.** (Optional) In the *Model destination* group box, specify a library, cell, and view name for the generated model, or click the *Browse* button to select the library, cell, and view name using the Select Model Destination form.



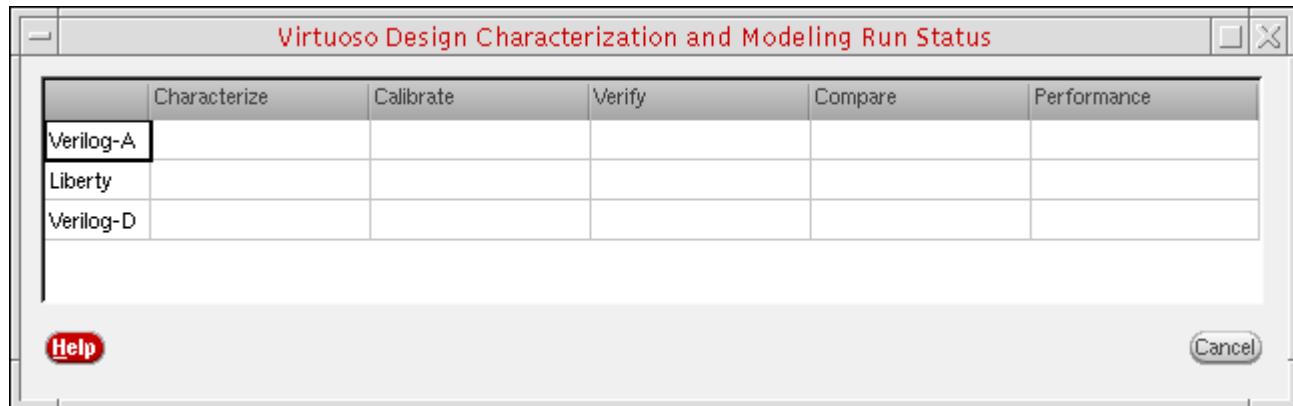
The program provides default library, cell, and view information. The default view is

- verilog\_a\_td for top-down Verilog-A[MS]
- verilog\_a for bottom-up Verilog-A[MS]
- verilog for bottom-up Verilog-D

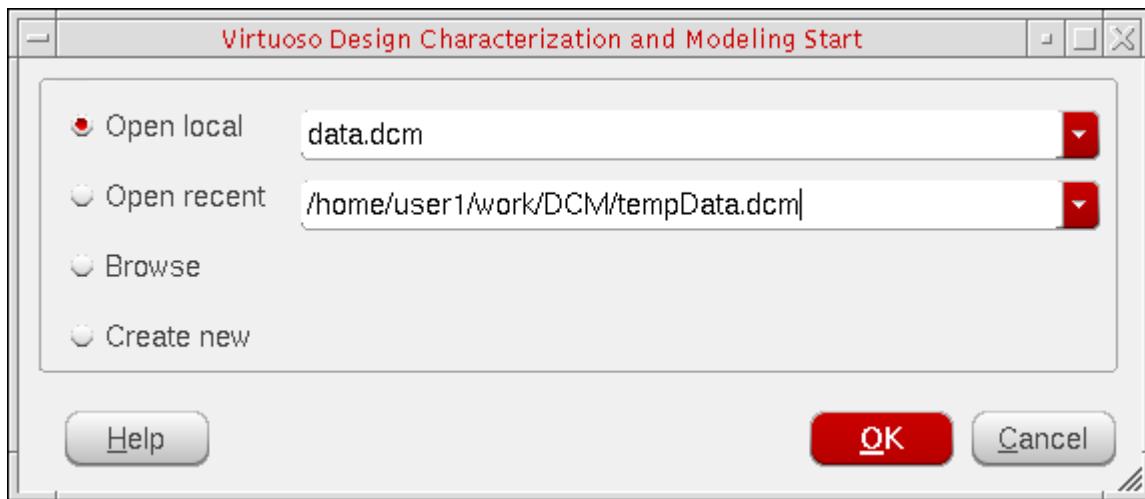
- 3.** Click *OK*.

The program copies the behavioral model to the specified library.

## Run Status



## Start



*Open local* lets you select a local data file (.dcm) located in the directory where you started the software.

*Open recent* lets you select a recently-opened data file.

*Browse* opens the Choose DCM Data File to Be Loaded form so that you can navigate to and select a data file to open.

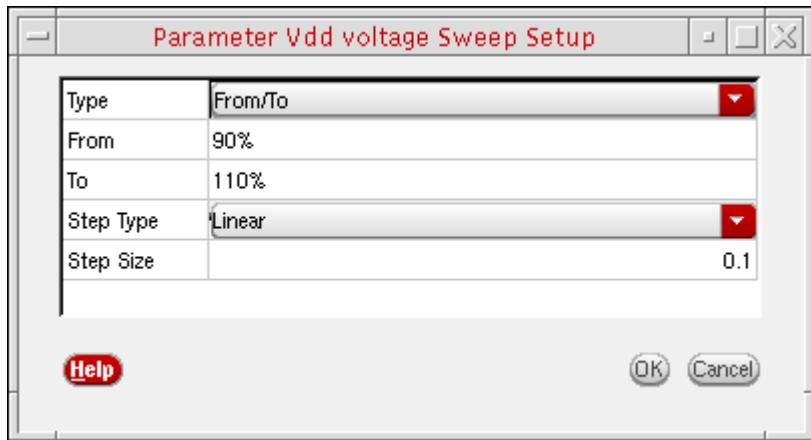
*Create new* opens the Virtuoso Design Characterization and Modeling window to the *Design* tab so that you can select a design.

*Help* opens the Online Help page for this form.

*OK* performs the task you selected and closes the form.

*Cancel* closes the form without performing any task.

## Sweep Setup



## Index

---

### A

adding  
    condition [374](#)  
    instance text [372](#)  
    measure [365](#)  
    model calibration statement [400](#)  
    models [398](#)  
    parameters [388](#)  
    test benches [357](#)  
addnig  
    pin rules [421](#)  
adel tests  
    dcmDeveloper wizard [341](#)  
adexl tests  
    dcmDeveloper wizard [338](#)  
analysis  
    sensitivity [20](#)

### B

bottom-up flow  
    dcmDeveloper wizard [326](#)  
Bottom-Up tab  
    model behaviors for Verilog-A [259](#)  
Bottom-Up tab (DCM)  
    Advanced button for Verilog-A  
        modeling [260](#)  
    Liberty tab [265](#)  
    Verilog-A tab [259](#)  
    Verilog-D tab [263](#)

### C

calibrating models [233, 269](#)  
    modifying sweep parameters [271](#)  
calibration  
    dcmDeveloper wizard [348](#)  
Calibration tab (DCM) [269](#)  
cell creation  
    options [416](#)  
cell type  
    information [416](#)  
    opening from DCM [426](#)

opening from dcmDeveloper [426](#)  
viewing [426](#)  
cell type generation  
    defaults [416](#)  
    destination [418](#)  
    options [414](#)  
    output [418](#)  
closing  
    dcmDeveloper [425](#)  
CML support [250](#)  
COMPONENT (OpenDCM) [456, 464](#)  
condition  
    adding [374](#)  
    language [375](#)  
    moving instances into [376](#)  
    moving models into [376](#)  
    pin type usage [374](#)  
configuring  
    test benches [357](#)  
context collisions [194](#)  
conventions  
    syntax [14](#)  
copy  
    test [362](#)  
correlation view  
    sensitivity data [61](#)  
creating  
    model [398](#)  
CSV  
    sensitivity data [95, 113](#)

### D

data  
    mismatch [108](#)  
data points [163](#)  
    definition [165](#)  
DCM  
    opening cell type [426](#)  
    running tests [283](#)  
DCM Bottom-Up tab  
    Liberty tab [265](#)  
    Verilog-A tab [259](#)  
    Verilog-D tab [263](#)  
DCM Calibration tab [269](#)

- 
- DCM Design tab [234](#)  
 DCM Design Verification tab [317](#)  
 DCM Function tab [241](#)  
 DCM Generate [282](#)  
 DCM Liberty tab [265](#)  
 DCM Options tab [281](#)  
 DCM Parameters tab [278](#)  
 DCM Top-Down tab [253](#)  
 DCM Verilog-A tab [259](#)  
 DCM Verilog-D tab [263](#)  
 DCM\_ANALOG (OpenDCM) [468](#), [470](#)  
 DCM\_BEGIN\_PERL (OpenDCM) [456](#),  
[470](#), [502](#)  
 DCM\_BEGIN\_PRIMITIVE  
 (OpenDCM) [456](#), [497](#)  
 DCM\_BITS (OpenDCM) [498](#)  
 DCM\_CALIBRATE (OpenDCM) [468](#), [471](#)  
 DCM\_CALIBRATE statement [400](#)  
 DCM\_COMMENT (OpenDCM) [468](#), [474](#)  
 DCM\_COUNT (OpenDCM) [484](#)  
 DCM\_DEBUG (OpenDCM) [468](#), [474](#)  
 DCM\_DEFINE (OpenDCM) [456](#), [462](#)  
 DCM DESIGN\_CELL (OpenDCM) [480](#)  
 DCM DESIGN\_LIBRARY  
 (OpenDCM) [480](#)  
 DCM DESIGN\_VIEW (OpenDCM) [480](#)  
 DCM\_ELSIF (OpenDCM) [456](#), [460](#)  
 DCM\_ERROR (OpenDCM) [456](#), [468](#), [497](#),  
[499](#)  
 DCM\_FOREACH (OpenDCM) [456](#), [459](#),  
[469](#)  
 nesting [457](#)  
 DCM\_GENERATE  
 adding [373](#)  
 DCM\_GENERATE (OpenDCM) [456](#), [457](#),  
[469](#)  
 nesting [457](#)  
 DCM\_GET\_VALUE (OpenDCM) [456](#), [497](#),  
[500](#)  
 DCM\_GUI environment variable [429](#)  
 DCM\_IF (OpenDCM) [456](#), [460](#), [469](#)  
 differential pins [462](#)  
 nesting [457](#)  
 DCM\_INV (OpenDCM) [481](#)  
 DCM\_IS\_PARAM\_SWEEP\_FROM\_TO\_BY  
 (OpenDCM) [485](#)  
 DCM\_IS\_PARAM\_SWEEP\_LIST  
 (OpenDCM) [485](#)  
 DCM\_IS\_PARAM\_SWEEP  
 (OpenDCM) [485](#)  
 DCM\_IS\_SIMULATOR (OpenDCM) [470](#)
- DCM\_IS\_SIMULATOR (OpenDCM) [497](#)  
 DCM\_LIB\_AREA (OpenDCM) [469](#), [489](#)  
 DCM\_LIB\_CALIBRATE\_DELAY  
 (OpenDCM) [469](#), [489](#), [490](#)  
 DCM\_LIB\_CALIBRATE\_HOLD  
 (OpenDCM) [469](#), [489](#), [491](#)  
 DCM\_LIB\_CALIBRATE\_INCAP  
 (OpenDCM) [469](#), [489](#), [491](#)  
 DCM\_LIB\_CALIBRATE\_LKGPWR  
 (OpenDCM) [469](#), [489](#), [492](#)  
 DCM\_LIB\_CALIBRATE\_NOCPWR  
 (OpenDCM) [469](#), [489](#), [492](#)  
 DCM\_LIB\_CALIBRATE\_RECOVERY  
 (OpenDCM) [469](#), [489](#), [493](#)  
 DCM\_LIB\_CALIBRATE\_SETUP  
 (OpenDCM) [469](#), [489](#), [494](#)  
 DCM\_LIB\_CALIBRATE\_SWIPWR  
 (OpenDCM) [469](#), [490](#)  
 DCM\_LIB\_CALIBRATE\_TRANSITION  
 (OpenDCM) [469](#), [490](#), [495](#)  
 DCM\_LIB\_CALIBRATE\_WIDTH  
 (OpenDCM) [469](#), [490](#), [496](#)  
 DCM\_LIB\_FUN (OpenDCM) [469](#), [489](#)  
 DCM\_LIB\_POWER (OpenDCM) [461](#), [489](#)  
 DCM LOWERCASE (OpenDCM) [497](#)  
 DCM\_MAX\_PIN\_ATT (OpenDCM) [483](#)  
 DCM\_MIN\_PIN\_ATT (OpenDCM) [483](#)  
 DCM\_MODULE (OpenDCM) [469](#), [475](#)  
 DCM\_NO\_MODULE (OpenDCM) [469](#),  
[475](#)  
 DCM\_ODCM\_LIB environment  
 variable [429](#)  
 DCM\_PARAM\_SWEEP\_BY  
 (OpenDCM) [485](#)  
 DCM\_PARAM\_SWEEP\_FROM  
 (OpenDCM) [485](#)  
 DCM\_PARAM\_SWEEP\_LIST  
 (OpenDCM) [485](#)  
 DCM\_PARAM\_SWEEP\_TO  
 (OpenDCM) [485](#)  
 DCM\_PARAM\_SWEEP (OpenDCM) [469](#),  
[487](#)  
 DCM\_PARAM\_VA (OpenDCM) [469](#), [486](#)  
 DCM\_PARAM\_VALUE (OpenDCM) [485](#)  
 DCM\_PIN\_ATT (OpenDCM) [483](#)  
 DCM\_PIN\_COUNT (OpenDCM) [484](#)  
 DCM\_PIN\_HIGH (OpenDCM) [482](#)  
 DCM\_PIN\_INDEX (OpenDCM) [484](#)  
 DCM\_PIN\_LOW (OpenDCM) [482](#)  
 DCM\_PIN\_THR (OpenDCM) [482](#)  
 DCM\_PINS\_LIST (OpenDCM) [484](#)

DCM\_PINS\_STRING (OpenDCM) [484](#)  
DCM\_SAME\_PIN (OpenDCM) [461](#)  
DCM\_SAVE\_SIGNAL (OpenDCM) [498](#),  
  [501](#)  
DCM\_SECTION (OpenDCM) [469](#), [475](#)  
DCM\_SIMULATOR (OpenDCM) [470](#), [497](#)  
DCM\_SKIP\_TEST (OpenDCM) [456](#), [464](#)  
DCM\_UPPERCASE (OpenDCM) [498](#)  
DCM\_V\_CONSTRAINTS (OpenDCM) [461](#)  
DCM\_V\_DEF\_DELAY (OpenDCM) [470](#),  
  [488](#)  
DCM\_V\_TIMING (OpenDCM) [470](#), [488](#)  
DCM\_VA\_DEF\_CAP (OpenDCM) [470](#)  
DCM\_VA\_DEF\_DELAY (OpenDCM) [470](#)  
DCM\_VA\_DEF\_SLOPE (OpenDCM) [470](#)  
DCM\_VA\_DYNAMIC\_VDD\_SAMPLE  
  (OpenDCM) [461](#)  
DCM\_VA\_MODULES (OpenDCM) [470](#),  
  [476](#)  
DCM\_VA\_VALUE (OpenDCM) [470](#), [498](#),  
  [501](#)  
DCM\_VARS (OpenDCM) [470](#), [476](#)  
DCM\_VDD (OpenDCM) [482](#)  
DCM\_VERIFY (OpenDCM) [470](#), [476](#)  
DCM\_VSS (OpenDCM) [482](#)  
DCM\_WAVE (OpenDCM) [498](#), [502](#)  
dcmDeveloper  
  adding measure [365](#)  
  closing [425](#)  
  copying test [362](#)  
  deleting test [363](#)  
  launching [328](#)  
  model calibration statement [400](#)  
  open cell type [426](#)  
  renaming test [363](#)  
  saving [425](#)  
  test benches [357](#)  
dcmDeveloper wizard  
  adel tests [341](#)  
  adexl tests [338](#)  
  bottom-up flow [326](#)  
  calibrating model [348](#)  
  design information [333](#)  
  design verification [327](#)  
  launching [331](#)  
  models [335](#)  
  objectives [331](#)  
  pin type rules [345](#)  
  pin types [343](#)  
  reviewing cell type [350](#)  
  tasks [352](#)

tests [337](#)  
top-down flow [326](#)  
understanding [325](#)  
viewing model [337](#)

delete  
  test [363](#)

deleting  
  parameter [393](#)  
  pin rule [424](#)  
  pin type [384](#)

design information  
  dcmDeveloper wizard [333](#)

design parameterization (for  
  optimization) [116](#)

design points [163](#)  
  definition [165](#)

Design tab (DCM) [234](#)

design verification [317](#)  
  defining specifications [318](#)  
  disabling tests and measurements [319](#)  
  enabling [317](#)  
  updating tests and measurements from  
    function information [319](#)

design verification flow  
  dcmDeveloper wizard [327](#)

Design Verification tab (DCM) [317](#)

device matching [116](#)

differential (!) pin notation  
  DCM\_IF (OpenDCM) [462](#)

differential input/output pin support [250](#)

## E

environment variables  
  DCM\_GUI [429](#)

## F

filtering  
  sensitivity data [72](#), [113](#)

Function tab (DCM) [241](#)

## G

Generate (DCM) [282](#)  
generating  
  Liberty MS model [287](#)

global optimization [134](#)

graphical view  
sensitivity data [60](#)

## I

improve yield [151](#)  
reference point [157](#)  
instance text  
adding [372](#)  
instances  
moving into condition [376](#)

## L

language condition [375](#)  
launching  
dcmDeveloper [328](#)  
dcmDeveloper wizard [331](#)  
Liberty MS  
model parameters [290](#)  
Liberty MS model  
generating [287](#)  
Liberty tab (DCM) [265](#)  
loading  
state files [396](#)  
local optimization [131](#)

## M

matching  
devices [116](#)  
properties [116](#)  
measure  
adding [365](#)  
message URL [../cdfuser/cdfuserTOC.html#firstpage](#) [125](#)  
mismatch data  
viewing [108](#)  
model  
component parameters [290](#)  
creating [398](#)  
options [404](#)  
viewing in wizard [337](#)  
model behavior  
options [406](#)  
model calibration [269](#)  
modifying sweep parameters [271](#)  
model calibration statement

adding [400](#)  
model generation [233](#)  
models  
adding [398](#)  
dcmDeveloper wizard [335](#)  
moving into condition [376](#)  
modifying  
parameters [391](#)  
pin rules [422](#)  
pin types [383](#)  
states [396](#)  
Monte Carlo  
mismatch data [108](#)  
yield improvement [151](#)  
Monte Carlo analysis  
Analysis Variation [24, 154, 170](#)  
statistical variation [24, 154, 170](#)  
Monte Carlo Sampling  
All [24, 154, 170](#)  
Mismatch [24, 154, 170](#)  
Process [24, 154, 170](#)  
moving  
instances into condition [376](#)  
models into condition [376](#)  
multi-technology simulation [185](#)

## O

objectives  
dcmDeveloper wizard [331](#)  
OpenDCM  
command reference [508](#)  
continuation character [468](#)  
function reference [508](#)  
macro reference [508](#)  
Perl extensions [502, 503](#)  
variables [508](#)  
[429](#)  
optimization  
design parameters for [116](#)  
global [134](#)  
local [131](#)  
reference point [157](#)  
resulting variables [136](#)  
running [130](#)  
yield improvement [151](#)  
options  
cell creation [416](#)  
cell type generation [414](#)  
model [404](#)

model behavior [406](#)  
Options tab (DCM) [281](#)  
output destination  
    cell type generation [418](#)  
overview  
    dcmDeveloper wizard [325](#)

**P**

parameterizing a design (for optimization) [116](#)  
parameters  
    adding [388](#)  
    deleting [393](#)  
    modifying [391](#)  
Parameters tab (DCM) [278](#)  
performance  
    specifications [122](#)  
Perl extensions (OpenDCM) [502](#)  
    dcm\_error [506](#)  
    dcm\_pin\_att [504](#)  
    dcm\_pin\_inv [504](#)  
    dcm\_pin\_rel [504](#)  
    debugging [506](#)  
    iterating through pin names [503](#)  
    special data access functions [504](#)  
pin rules  
    adding [421](#)  
    deleting [424](#)  
    modifying [422](#)  
pin type rules  
    dcmDeveloper wizard [345](#)  
pin type usage condition [374](#)  
pin types  
    dcmDeveloper wizard [343](#)  
    deleting [384](#)  
    iteration [373](#)  
    modifying [383](#)  
points  
    design points [165](#)  
    saving data [163](#)  
points data points [165](#)  
properties  
    matching [116](#)  
    varying [116](#)

## R

ratio-matching

properties [116](#)  
reference point  
    for optimization [157](#)  
    modifying [159](#)  
    viewing for run [159](#)  
regression view  
    sensitivity data [66, 67](#)  
rename  
    test [363](#)  
reviewing cell type  
    dcmDeveloper wizard [350](#)  
run  
    view reference point [159](#)  
running  
    optimization [130](#)

**S**

saving  
    dcmDeveloper data [425](#)  
    sensitivity data [95, 113](#)  
sensitivity data  
    converting to CSV [95, 113](#)  
    correlation view [61](#)  
    data type [60](#)  
    filtering [72, 113](#)  
    regression view [66, 67](#)  
    saving [95, 113](#)  
    view [60](#)  
    viewing [20](#)  
sensitivity data  
    graphical view [60](#)  
simulation  
    multi-technology [185](#)  
SKILL Name Checker [194](#)  
SKILL name collisions [194](#)  
specifications  
    setting up [122](#)  
specifications (for design verification) [318](#)  
    disabling tests and measurements [319](#)  
    Update from function [319](#)  
state files  
    loadnig [396](#)  
states  
    modifying [396](#)  
syntax  
    conventions [14](#)

## T

tasks  
  dcmDeveloper wizard [352](#)  
test  
  copying [362](#)  
  deleting [363](#)  
  renaming [363](#)  
test benches  
  adding [357](#)  
tests  
  dcmDeveloper wizard [337](#)  
top-down flow  
  dcmDeveloper wizard [326](#)  
top-down modeling  
  specifying advanced options for [256](#)  
  specifying design pin types for [242](#)  
  specifying functional cell type for [241](#)  
  specifying model options for [255](#)  
  specifying parameter values for [255](#)  
  specifying the simulator for [243](#)  
  Top-Down tab [253](#)  
Top-Down tab (DCM) [253](#)

View button in [213](#)

## Y

yield  
  improve [151](#)

## U

Update from function [319](#)  
user-defined primitives [264](#)

## V

variables  
  optimization results [136](#)  
  setting up [116](#)  
verify design to specification [317](#)  
Verilog-A tab (DCM) [259](#)  
Verilog-D model  
  user-defined primitives [264](#)  
Verilog-D tab (DCM) [263](#)  
view  
  sensitivity data [60](#)  
View menu (DCM) [212](#)  
  Last Error [211](#)  
viewing  
  cell type [426](#)  
  mismatch data [108](#)  
  sensitivity data [20](#)  
viewing window (DCM) [212](#)