Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm
Abstract

This manual describes the use of the Xyce Parallel Electronic Simulator. Xyce has been designed as a SPICE-compatible, high-performance analog circuit simulator, and has been written to support the simulation needs of the Sandia National Laboratories electrical designers. This development has focused on improving capability over the current state-of-the-art in the following areas:

- Capability to solve extremely large circuit problems by supporting large-scale parallel computing platforms (up to thousands of processors). Note that this includes support for most popular parallel and serial computers.
- Improved performance for all numerical kernels (e.g., time integrator, nonlinear and linear solvers) through state-of-the-art algorithms and novel techniques.
- Device models which are specifically tailored to meet Sandia’s needs, including many radiation-aware devices.
Object-oriented code design and implementation using modern coding practices that ensure that the Xyce Parallel Electronic Simulator will be maintainable and extensible far into the future.

Xyce is a parallel code in the most general sense of the phrase - a message passing parallel implementation - which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved as the number of processors grows.

The development of Xyce provides a platform for computational research and development aimed specifically at the needs of the Laboratory. With Xyce, Sandia has an “in-house” capability with which both new electrical (e.g., device model development) and algorithmic (e.g., faster time-integration methods, parallel solver algorithms) research and development can be performed. As a result, Xyce is a unique electrical simulation capability, designed to meet the unique needs of the laboratory.
Acknowledgements

The authors would like to acknowledge the entire Sandia National Laboratories HPEMS (High Performance Electrical Modeling and Simulation) team, including Steve Wix, Carolyn Bogdan, Regina Schells, Ken Marx, Steve Brandon and Bill Ballard, for their support on this project. We also appreciate very much the work of Jim Emery, Becky Arnold and Mike Williamson for the help in reviewing this document.

Lastly, a very special thanks to Hue Lai for typesetting this document with \LaTeX.

Trademarks

The information herein is subject to change without notice.

Copyright © 2002-2003 Sandia Corporation. All rights reserved.

Xyce™ Electronic Simulator and Xyce™ trademarks of Sandia Corporation.

Orcad, Orcad Capture, PSpice and Probe are registered trademarks of Cadence Design Systems, Inc.

Silicon Graphics, the Silicon Graphics logo and IRIX are registered trademarks of Silicon Graphics, Inc.

Microsoft, Windows and Windows 2000 are registered trademark of Microsoft Corporation.

Solaris and UltraSPARC are registered trademarks of Sun Microsystems Corporation.

Medici, DaVinci and Taurus are registered trademarks of Synopsys Corporation.

HP and Alpha are registered trademarks of Hewlett-Packard company.

Amtec and TecPlot are trademarks of Amtec Engineering, Inc.

Xyce's expression library is based on that inside Spice 3F5 developed by the EECS Department at the University of California.

All other trademarks are property of their respective owners.

Contacts

Bug Reports http://tvrusso.sandia.gov/bugzilla
Email xyce-support@sandia.gov
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>15</td>
</tr>
<tr>
<td>1.1 Xyce Overview</td>
<td>15</td>
</tr>
<tr>
<td>1.2 Xyce Capabilities</td>
<td>16</td>
</tr>
<tr>
<td>Support for large-scale parallel computing</td>
<td>16</td>
</tr>
<tr>
<td>Improved performance for all numerical kernels</td>
<td>16</td>
</tr>
<tr>
<td>Device Model Support</td>
<td>17</td>
</tr>
<tr>
<td>1.3 Description of Document</td>
<td>18</td>
</tr>
<tr>
<td>1.4 Reference Guide</td>
<td>19</td>
</tr>
<tr>
<td>1.5 How to Use this Guide</td>
<td>19</td>
</tr>
<tr>
<td>2. Installing and Running Xyce</td>
<td>21</td>
</tr>
<tr>
<td>2.1 Xyce Installation</td>
<td>21</td>
</tr>
<tr>
<td>Installing Xyce on UNIX</td>
<td>22</td>
</tr>
<tr>
<td>Installing Xyce on Microsoft Windows</td>
<td>22</td>
</tr>
<tr>
<td>Important Notes</td>
<td>31</td>
</tr>
<tr>
<td>2.2 Running Xyce</td>
<td>32</td>
</tr>
<tr>
<td>Command Line Simulation</td>
<td>32</td>
</tr>
<tr>
<td>Command Line Options</td>
<td>33</td>
</tr>
<tr>
<td>Running Xyce in Parallel</td>
<td>34</td>
</tr>
<tr>
<td>Accessing the Microsoft Windows Command Line</td>
<td>35</td>
</tr>
<tr>
<td>3. Simulation Examples with Xyce</td>
<td>41</td>
</tr>
<tr>
<td>3.1 Example Circuit Construction</td>
<td>41</td>
</tr>
<tr>
<td>3.2 DC Sweep Analysis</td>
<td>42</td>
</tr>
<tr>
<td>3.3 Transient Analysis</td>
<td>45</td>
</tr>
<tr>
<td>4. Netlist Basics</td>
<td>51</td>
</tr>
<tr>
<td>4.1 General Overview</td>
<td>51</td>
</tr>
<tr>
<td>Introduction</td>
<td>52</td>
</tr>
<tr>
<td>Netlist Elements</td>
<td>52</td>
</tr>
<tr>
<td>4.2 Devices Available for Simulation</td>
<td>55</td>
</tr>
<tr>
<td>Analog Devices</td>
<td>55</td>
</tr>
<tr>
<td>4.3 Parameters and Expressions</td>
<td>56</td>
</tr>
<tr>
<td>Parameters</td>
<td>57</td>
</tr>
<tr>
<td>How to Declare and Use Parameters</td>
<td>57</td>
</tr>
</tbody>
</table>
## Expressions

58

### 5. Working with .MODEL Statements and Subcircuit Models

61

#### 5.1 Definition of a Model

62
- Defining models using model parameters
- Defining models using subcircuit netlists

#### 5.2 Model Organization

65
- Model libraries
- Model library configuration

### 6. Analog Behavioral Modeling

67

#### 6.1 Overview of Analog Behavioral Modeling

68

#### 6.2 Specifying ABM Devices

68
- Additional constructs for use in ABM expressions
- Lookup Tables in Analog Behavioral Modeling
- Alternate behavioral modeling sources

#### 6.3 Guidance for ABM Use

70

### 7. Analysis Types

73

#### 7.1 Introduction

74

#### 7.2 Transient Analysis

74
- .TRAN Statement
- Defining a Time-Dependent (transient) Source
- Transient Calculation Time Steps
- Checkpointing and Restarting

#### 7.3 DC Analysis

77
- .DC Statement
- Setting Up and Running a DC Sweep
- OP Analysis

#### 7.4 STEP Parametric Analysis

81
- .STEP Statement
- Sweeping over a Device Instance Parameter
- Sweeping over a Device Model Parameter
- Sweeping over Temperature
- Special cases: Sweeping Independent Sources, Resistors, Capacitors
- Output files

### 8. Using Homotopy Algorithms to Obtain Operating Points

87

#### 8.1 Homotopy Algorithms Overview

88
- HOMOTOPY Algorithms Available in Xyce

#### 8.2 Examples

88
- MOSFET Homotopy
- Natural Parameter Homotopy

### 9. Results Output and Evaluation Options

93

#### 9.1 Control of Results Output

94
CONTENTS

10. Guidance for Running Xyce in Parallel
   10.1 Introduction ................................................................. 100
   10.2 Mechanics ................................................................. 100
   10.3 Problem Size ............................................................... 100
      Smallest Possible Problem Size ........................................... 100
      Ideal Problem Size ....................................................... 100
   10.4 Linear Solver Options .................................................... 101
   10.5 Partitioning Options ..................................................... 101
      Zoltan Partitioning of the Linear System ............................... 102
      Singleton Filtering of the Linear System .............................. 102

11. TCAD (PDE Device) Simulation with Xyce
   11.1 Introduction ............................................................... 104
   11.2 One Dimensional Example ............................................... 107
      Netlist Explanation ....................................................... 108
      Boundary Conditions and Doping Profile ............................... 109
      Results ................................................................. 109
   11.3 Two Dimensional Example ............................................... 111
      Netlist Explanation ....................................................... 111
      Doping Profile .......................................................... 113
      Boundary Conditions and Electrode Configuration .................... 113
      Results ................................................................. 113
   11.4 Doping Profile ............................................................. 118
      Manually Specifying the Doping .......................................... 118
      Default Doping Profiles ................................................ 121
   11.5 Electrodes ................................................................. 123
      Manually Specifying the Electrodes ..................................... 123
      Electrode Defaults ........................................................ 126
   11.6 Meshes ................................................................. 127
      Meshes from the SG Framework (External, 2D) ......................... 127
      Cartesian Meshes (Internal, 1D and 2D) .............................. 127
      Cylindrical meshes, 2D .................................................. 127
   11.7 Mobility Models .......................................................... 129
   11.8 Bulk Materials .......................................................... 130
   11.9 Solver Options ........................................................... 131
   11.10 Output and Visualization ............................................... 132
      Using the .PRINT Command ............................................... 132
      Multi-dimensional Plots ................................................ 132
      Volume Averaged Data ................................................... 133

Using the .PRINT Command ................................................ 94

9.2 Additional Output Options .............................................. 94

9.3 Evaluating Solution Results ........................................... 96

8. Mechanics ..................................................................... 100
   8.1 Introduction ................................................................. 100
   8.2 Netlist Explanation ....................................................... 104
   8.3 Results ....................................................................... 106

7. Discretization .................................................................. 104
   7.1 Introduction ................................................................. 104
   7.2 Equations .................................................................... 106
   7.3 Results ....................................................................... 106

6. Boundary Conditions and Doping Profile ............................... 109
   6.1 Introduction ................................................................. 109
   6.2 Results ....................................................................... 109

5. Results ........................................................................... 109
   5.1 Introduction ................................................................. 109
   5.2 Equations .................................................................... 110
   5.3 Results ....................................................................... 111

4. Boundary Conditions and Electrode Configuration .................... 113
   4.1 Introduction ................................................................. 113
   4.2 Results ....................................................................... 115

3. Doping Profile ................................................................... 118
   3.1 Introduction ................................................................. 118
   3.2 Results ....................................................................... 120

2. Electrodes ....................................................................... 123
   2.1 Introduction ................................................................. 123
   2.2 Results ....................................................................... 126

1. Meshes ........................................................................... 127
   1.1 Introduction ................................................................. 127
   1.2 Results ....................................................................... 129

PRINT Command ........................................................... 94

OPTIONS OUTPUT Command .................................................... 94

TCAD (PDE Device) Simulation with Xyce ................................. 103

Users' Guide
This page is left intentionally blank
Figures

2.1 Install Xyce windows.zip Icon .................................................. 23
2.2 Install Xyce windows.zip Contents ............................................. 24
2.3 Extraction Wizard: Welcome ....................................................... 25
2.4 Extraction Wizard: Select Destination .......................................... 26
2.5 Extraction Wizard: Show Extracted Files ....................................... 27
2.6 Contents of Extracted Archive .................................................... 28
2.7 Xyce Setup Icon ........................................................................... 29
2.8 On-screen Setup Instructions ......................................................... 30
2.9 Status Messages and Success Confirmation .................................... 31
2.10 Using the Start Menu ................................................................. 35
2.11 Using the Run Dialog Box ............................................................ 36
2.12 Default Command Line Window .................................................. 36
2.13 Command Line Window: Using runxyce ....................................... 37
2.14 Command Line Window: Default runxyce Output ......................... 38
2.15 Command Line Window: Starting a Simulation ............................... 39
2.16 Command Line Window: On-screen Output ................................... 40

3.1 Schematic of diode clipper circuit with DC and transient voltage sources ........................................... 43
3.2 Diode clipper circuit netlist ........................................................... 44
3.3 DC sweep voltages at Vin, node 2 and Vout ..................................... 45
3.4 Diode clipper circuit netlist for DC sweep analysis ......................... 46
3.5 Diode clipper circuit netlist for transient analysis ......................... 48
3.6 Sinusoidal input signal and clipped outputs ..................................... 49

5.1 Example subcircuit model ............................................................... 63
5.2 Example subcircuit model ............................................................... 64

7.1 Diode clipper circuit netlist for DC sweep analysis .......................... 79
7.2 DC sweep voltages at Vin, node 2 and Vout ..................................... 80
7.3 Diode clipper circuit netlist for step transient analysis .................... 83
7.4 Diode clipper circuit netlist for 2-step transient analysis ................. 84

8.1 Example MOSFET homotopy netlist ............................................. 89
8.2 Example natural parameter homotopy netlist .................................. 91

9.1 TecPlot plot of diode clipper circuit transient response from Xyce .prn file ......................................................... 96
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>MOSFET Mesh Example</td>
<td>105</td>
</tr>
<tr>
<td>11.2</td>
<td>One dimensional diode netlist</td>
<td>107</td>
</tr>
<tr>
<td>11.3</td>
<td>Voltage regulator schematic</td>
<td>108</td>
</tr>
<tr>
<td>11.4</td>
<td>Transient Result for voltage regulator</td>
<td>110</td>
</tr>
<tr>
<td>11.5</td>
<td>Two-dimensional BJT netlist</td>
<td>112</td>
</tr>
<tr>
<td>11.6</td>
<td>Two-Dimensional BJT Circuit Schematic</td>
<td>115</td>
</tr>
<tr>
<td>11.7</td>
<td>Initial Two-Dimensional BJT Result</td>
<td>116</td>
</tr>
<tr>
<td>11.8</td>
<td>Final Two-Dimensional BJT Result</td>
<td>116</td>
</tr>
<tr>
<td>11.9</td>
<td>I-V Two-Dimensional BJT Result</td>
<td>117</td>
</tr>
<tr>
<td>11.10</td>
<td>One-dimensional example, with detailed doping</td>
<td>119</td>
</tr>
<tr>
<td>11.11</td>
<td>Doping Profile</td>
<td>120</td>
</tr>
<tr>
<td>11.12</td>
<td>Two-dimensional example, with detailed doping and detailed electrodes</td>
<td>124</td>
</tr>
<tr>
<td>11.13</td>
<td>Cylindrical Mesh Example</td>
<td>128</td>
</tr>
</tbody>
</table>
This page is left intentionally blank
1. Introduction

Welcome to Xyce

The Xyce Parallel Electronic Simulator is a SPICE-compatible [1] [2] circuit simulator, that has been written to support the unique simulation needs of electrical designers at Sandia National Laboratories. It is targeted specifically to run on large-scale parallel computing platforms but is also available on a variety of architectures including single processor workstations. It aims to support a variety of devices and models specific to Sandia needs as well as the standard capability exhibited by current commercial simulators.
1.1 **Xyce** Overview

The **Xyce** Parallel Electronic Simulator project was started in 1999 to support the simulation needs of electrical designers at Sandia National Laboratories. The current release of **Xyce** is version 2.1, and the code has evolved into a mature platform for large scale circuit simulation.

**Xyce** includes several unique features. In addition to allowing the simulation of circuits of unprecedented size, **Xyce** includes novel approaches to numerical kernels including time integration algorithms, nonlinear and linear solvers. The primary driver for this numerical innovation has been the need to simulate very large scale circuits (100,000 devices or more) on the analog level. However, it has yielded benefits, in terms of robustness and efficiency, for all classes of problems. Ideally, the increased numerical robustness minimizes the amount of simulation “tuning” required on the part of the designer.

1.2 **Xyce** Capabilities

**Xyce** has a number of unique features which are described in this section.

**Support for large-scale parallel computing**

**Xyce** is a truly parallel simulation code, designed and written from the ground up to support large-scale (up to thousands of processors) parallel computing architectures. This gives **Xyce** the capability to solve circuit problems of unprecedented size in time frames that make these simulations practical.

**Xyce** as a parallel code uses a message passing parallel implementation, which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel. Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows (*parallel scaling*).

**Improved performance for all numerical kernels**

In writing **Xyce** from scratch, new algorithms and heuristics have been used which improve the overall performance of the various numerical kernels. For example, a number of new developments have made it possible to reliably apply iterative linear solvers to circuit problems. This allows **Xyce** to scale well to much larger problem sizes than would be possible with a conventional circuit simulator. Using iterative linear solvers also allows **Xyce** to run much more effectively in parallel.
On the nonlinear solver level, the addition of continuation algorithms to Xyce has been another recent solver enhancement. In particular, Xyce has been very successful applying such algorithms to large MOSFET circuits. See chapter [8] for more details.

Device Model Support

New device models are continually being added to Xyce, to meet the needs of Sandia users. Table [1.1] contains a complete list of devices for Xyce Release 2.1. For a complete description of each device, see the Xyce Reference Guide [3]. As there are many devices under development, several devices are available in the development branch of the code that are not available in the release branch. For current device availability, consult with the Xyce development team.

<table>
<thead>
<tr>
<th>Device</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitor</td>
<td>Age-aware, semiconductor</td>
</tr>
<tr>
<td>Inductor</td>
<td>Nonlinear mutual inductor (see below)</td>
</tr>
<tr>
<td>Nonlinear Mutual Inductor</td>
<td>Sandia Core model (not fully PSpice compatible)</td>
</tr>
<tr>
<td>Resistor</td>
<td>Semiconductor</td>
</tr>
<tr>
<td>Diode (Level 1)</td>
<td></td>
</tr>
<tr>
<td>Diode (Level 3)</td>
<td>Prompt photocurrent radiation model</td>
</tr>
<tr>
<td>Independent Voltage Source (VSRC)</td>
<td></td>
</tr>
<tr>
<td>Independent Current Source (ISRC)</td>
<td></td>
</tr>
<tr>
<td>Voltage Controlled Voltage Source (VCVS)</td>
<td></td>
</tr>
<tr>
<td>Voltage Controlled Current Source (VCCS)</td>
<td></td>
</tr>
<tr>
<td>Current Controlled Voltage Source (CCVS)</td>
<td></td>
</tr>
<tr>
<td>Voltage Controlled Current Source (CCCS)</td>
<td></td>
</tr>
<tr>
<td>Nonlinear Dependent Source (B Source)</td>
<td></td>
</tr>
<tr>
<td>Bipolar Junction Transistor (BJT)(Level 1)</td>
<td></td>
</tr>
<tr>
<td>Bipolar Junction Transistor (BJT)(Level 2)</td>
<td>Prompt photocurrent radiation model.</td>
</tr>
<tr>
<td>Bipolar Junction Transistor (BJT)(Level 3)</td>
<td>New! Neutron-effects model.</td>
</tr>
<tr>
<td>Bipolar Junction Transistor (BJT)(Level 4)</td>
<td>New! Alternate prompt photocurrent radiation model.</td>
</tr>
</tbody>
</table>
### Table 1.1: Devices Supported by Xyce.

<table>
<thead>
<tr>
<th>Device</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Junction Field Effect Transistor (JFET)</td>
<td>New! Level-1 JFET model.</td>
</tr>
<tr>
<td>MOSFET (Level 1)</td>
<td></td>
</tr>
<tr>
<td>MOSFET (Level 3)</td>
<td></td>
</tr>
<tr>
<td>MOSFET (Level 9)</td>
<td>BSIM3 model.</td>
</tr>
<tr>
<td>MOSFET (Level 18)</td>
<td>New! VDMOS model.</td>
</tr>
<tr>
<td>MOSFET (Level 19)</td>
<td>New! VDMOS photocurrent model.</td>
</tr>
<tr>
<td>Transmission Line</td>
<td>Lossless.</td>
</tr>
<tr>
<td>Voltage Controlled Switch (VSWITCH)</td>
<td></td>
</tr>
<tr>
<td>PDE Devices (Level 1)</td>
<td>one-dimensional</td>
</tr>
<tr>
<td>PDE Devices (Level 2)</td>
<td>two-dimensional</td>
</tr>
</tbody>
</table>

1.3 **Description of Document**

For the user, this document contains a description of the Xyce Parallel Electronic Simulator, in which the following topics are specifically addressed. Chapter 2 gives a quick-start guide to installing and running Xyce. Chapter 3 gives some examples of using the code. Chapters 4 through 7 describe the netlist language and its usage in Xyce. Chapter 8 covers the use of continuation algorithms, which is a unique capability for a circuit code such as Xyce. This is followed by Chapter 9 that covers analyzing the results output by the code. Chapter 10 provides guidance for running Xyce in parallel. The final chapter, Chapter 11 described the usage of mesh-based TCAD devices, which are based on solving a set of discretized partial differential equations (PDE) on a mesh, similar to commercial device simulators such as Medici.
1.4 Reference Guide

A companion document, the Xyce Reference Guide [3], contains more detailed information about a number of topics. Included in this document is a netlist reference for the input-file commands and elements supported within Xyce; a command line reference, which describes the available command line arguments for Xyce; and quick-references for users of other circuit codes, such as Orcad’s PSpice [4] and Sandia’s ChileSPICE.

1.5 How to Use this Guide

This guide is designed so you can quickly find the information you need to use Xyce. It assumes that you are familiar with basic Unix-type commands, how Unix manages applications and files to perform routine tasks (e.g., starting applications, opening files and saving your work).

Typographical conventions

Before continuing in this Users’ Guide, it is important to understand the terms and typographical conventions used. Procedures for performing an operation are generally numbered with the following typographical conventions.
<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verbatim text</td>
<td>&gt; xmpirun -np 4</td>
<td>Commands entered from the keyboard on the command line or text entered in a netlist. The initial &gt; character is intended to represent the shell prompt.</td>
</tr>
<tr>
<td>Bold Roman Font</td>
<td>Set nominal temperature using the TNOM option.</td>
<td>SPICE-type parameters used in models, etc.</td>
</tr>
<tr>
<td>Gray Shaded Text</td>
<td>DEBUGLEVEL</td>
<td>Feature that is designed primarily for use by Xyce developers.</td>
</tr>
<tr>
<td>[text in brackets]</td>
<td>Xyce [options] &lt;netlist&gt;</td>
<td>Optional parameters.</td>
</tr>
<tr>
<td>&lt;text in angle brackets&gt;</td>
<td>Xyce [options] &lt;netlist&gt;</td>
<td>Parameters to be inserted by the user.</td>
</tr>
<tr>
<td>&lt;object with asterisk&gt;*</td>
<td>K1 &lt;ind. 1&gt; [&lt;ind. n&gt;*]</td>
<td>Parameter that may be multiply specified.</td>
</tr>
<tr>
<td>&lt;TEXT1</td>
<td>TEXT2&gt;</td>
<td>.PRINT TRAN + DELIMITER=&lt;TAB</td>
</tr>
</tbody>
</table>

Table 1.2. Xyce typographical conventions.
2. Installing and Running Xyce

Chapter Overview

This chapter describes the basic mechanics of installing and running Xyce. It includes the following sections:

- Section 2.1 Xyce Installation
- Section 2.2 Running Xyce
2.1 Xyce Installation

To obtain a copy of Xyce, contact the development team at http://www.cs.sandia.gov/xyce.

The installation procedure differs depending upon the operating system. Follow the instructions below to properly install Xyce.

Installing Xyce on UNIX

Xyce is installed from the command line. Users of Linux, BSD, IRIX, and other UNIX variants should follow this section. Examples are given for reference.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation packages are named according the target operating system and</td>
<td>Install_Xyce_linux.tar.gz (serial)</td>
</tr>
<tr>
<td>architecture (parallel or serial).</td>
<td>Install_Xyce_linux_MPI.tar.gz (parallel)</td>
</tr>
<tr>
<td>Unpack the appropriate package for your platform. A similarly named</td>
<td>&gt; gzip -d Install_Xyce_linux.tar.gz</td>
</tr>
<tr>
<td>installation directory is then created.</td>
<td>&gt; tar xf Install_Xyce_linux.tar</td>
</tr>
<tr>
<td>Enter this directory and run the installation shell script.</td>
<td>&gt; cd Install_Xyce_linux</td>
</tr>
<tr>
<td></td>
<td>&gt; sh install_linux.sh</td>
</tr>
<tr>
<td>Provide the requested information.</td>
<td>Where should Xyce be installed?</td>
</tr>
<tr>
<td></td>
<td>/usr/local/Xyce-2.1</td>
</tr>
</tbody>
</table>

Installing Xyce on Microsoft Windows

The installation package is named Install_Xyce_windows.zip. This example illustrates the procedure for Windows XP Professional. Similar steps may be followed for other versions of Windows.
Figure 2.1. Obtain a copy of the installation package. Double click the file name to begin.
Figure 2.2. By default, Windows will display the contents of the zip file as shown above. Select Extract all Files to continue.

Users of programs such as WinZip, PKZip, Winrar, etc., may follow the decompression steps associated with those programs, and continue from Figure 2.6.
2.1 Xyce Installation

Figure 2.3. Click Next on the Extraction Wizard.
Figure 2.4. Select a destination for the archive contents and click Next.
Figure 2.5. Select **Show extracted files** and click **Finish**.
Figure 2.6. The contents of the extracted zip file are shown. Open the folder named **Install_Xyce_windows**.
Figure 2.7. Double click Setup to begin.
Figure 2.8. A command line window will open. Follow the on-screen instructions to proceed.
2.1 Xyce Installation

Figure 2.9. Status messages will display during setup. Confirmation of success will appear once all components are installed.

Important Notes

Completing the steps above will unpack Xyce to the specified directory. IMPORTANT NOTE: if installing both serial and parallel versions of Xyce, you must specify different directories for each installation location. Failure to use different directories will cause the second installation to overwrite parts of the first and will likely yield an install that does not function. Under the specified installation directories, the following subdirectories will be created:

- **bin** contains the executable used to start Xyce. The executable name will vary depending on the target operating system and architecture.
  
  - runxyce is the shell script for starting serial Xyce on Unix platforms.
  - runxyce.bat is the batch file for starting serial Xyce on Windows.
  - xmpirun is the wrapper script for mpirun used for running Xyce in parallel mode.
2.2 Running Xyce

ChileCAD [5], a GUI for Xyce is under development and will eventually become part of the overall simulation framework. This section only describes how Xyce is run from the command line, for both serial and MPI parallel simulations.

Command Line Simulation

Running Xyce from the command line is straightforward. The scripts xmpirun and runxyce set up the runtime environment and execute Xyce. Help with accessing the command line on Microsoft Windows is available at the end of this chapter. Depending on whether you are using a version compiled with MPI support or a serial version, there are two ways to begin running Xyce:

- Running serial Xyce:
  - > runxyce [options] <netlist filename>

- Running Xyce in parallel:
  - > xmpirun -np <# procs> [options] <netlist filename>

where [options] are the command line arguments for Xyce. For example, to log output to a file named sample.log type:

  - > runxyce -l sample.log <netlist filename>

The next example runs parallel Xyce on four processors and places the results into a comma separated value file named results.csv:

  - > xmpirun -np 4 -delim COMMA -o results.csv <netlist filename>
These examples assume that `<netlist filename>` is either in the current working directory or includes the path (full or relative) to the netlist file. Enclose the filename in quotation marks (" ") if the path contains spaces. Help is accessible with the `-h` option.

For MPI runs, `[options]` may also include command line arguments to `mpirun`. Consult the documentation installed with MPI on your platform for more details on MPI options. The `-np <# procs>` denotes the number of processors to use for the simulation. **NOTE: It is critical that the number of processors used is less than the number of devices and voltage nodes in the netlist.** The appropriate script used to run `Xyce` for each supported platform is listed in the Table 2.1.

<table>
<thead>
<tr>
<th>Computer Architecture</th>
<th>OS</th>
<th>Serial Executable</th>
<th>MPI Executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple PPC</td>
<td>OSX</td>
<td>runxyce</td>
<td>Not Available</td>
</tr>
<tr>
<td>HP/Compaq</td>
<td>Tru64</td>
<td></td>
<td>xmpirun</td>
</tr>
<tr>
<td>SGI 64 bit</td>
<td>IRIX 6.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel X86</td>
<td>Linux</td>
<td>runxyce.bat</td>
<td></td>
</tr>
<tr>
<td>Intel X86</td>
<td>FreeBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel X86</td>
<td>Microsoft Windows XP</td>
<td>runxyce.bat</td>
<td>Not Available</td>
</tr>
</tbody>
</table>

**Table 2.1.** Platform scripts for running `Xyce`.

While `Xyce` is running, the progress of the simulation is output to the command line window.

**Command Line Options**

`Xyce` supports a handful of command line options which must be given before the netlist filename. The general usage is:

```
runxyce [options] <netlist filename>
```

Table 2.2 lists the available command line options.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Usage</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Help option. Prints usage and exits.</td>
<td>-h</td>
<td>-</td>
</tr>
<tr>
<td>-v</td>
<td>Prints the version banner and exits.</td>
<td>-v</td>
<td>-</td>
</tr>
</tbody>
</table>
### Xyce™ Users' Guide

## Installing and Running Xyce

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Usage</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-delim</td>
<td>Set the output file field delimiter.</td>
<td>-delim &lt;TAB</td>
<td>COMMA</td>
</tr>
<tr>
<td>-o</td>
<td>Place the results into specified file.</td>
<td>-o &lt;file&gt;</td>
<td>-</td>
</tr>
<tr>
<td>-l</td>
<td>Place the log output into specified file.</td>
<td>-l &lt;file&gt;</td>
<td>-</td>
</tr>
<tr>
<td>-nox</td>
<td>Use the NOX nonlinear solver.</td>
<td>-nox &lt;on</td>
<td>off&gt;</td>
</tr>
<tr>
<td>-linsolv</td>
<td>Set the linear solver.</td>
<td>-linsolv &lt;ksparse</td>
<td>klu</td>
</tr>
</tbody>
</table>

Table 2.2: List of Xyce command line arguments.

While these options are intended for general use, others may exist for new features that are disabled by default, and older features that are no longer supported. See the Xyce Reference Guide for a comprehensive list that also includes trial and deprecated options.

### Running Xyce in Parallel

A parallel version of Xyce is available for several different platforms as shown in Table 2.1. Running Xyce in parallel requires the script xmpirun to be used with the appropriate parameters. For example, to run Xyce on two processors with an example netlist, type:

```
xmpirun -np 2 anExampleNetlist.cir
```

In general the number of processors is specified by using the `-np` argument to the appropriate mpirun command. Some specific considerations are given below.

### Running Xyce under MPICH

The MPICH implementation of MPI requires that there exist a file of machines on which to run. On RedHat Linux this is installed in `/usr/lib/mpich/share`. On FreeBSD this is installed in `/usr/local/mpich/share`. This file must contain one line for each machine on which a process may be started. If you do not have write access to the directory in which the default machines file is stored you may specify an alternate file with the `-machinefile <machinefilename>` option to mpirun.

MPICH executes parallel jobs by using the remote shell (rsh) or secure shell (ssh) to the target machine. You might, therefore, be prompted for a password when starting up a multiple processor job.
2.2 Running Xyce

Running Xyce under LAM MPI

Unlike MPICH, LAM MPI requires a daemon process to be running on each machine that will service parallel jobs. This daemon is started by using the lamboot program. By default, lamboot will run a daemon on the local machine, but it may be given a file name containing a list of machines for multiple-machine jobs. Consult the bhost man page for the format of the file.

lamboot runs a program called lamd which will remain running until it is halted. As long as lamd is running you may continue to run parallel jobs. Halt lamd using the lamhalt command.

Guidance

This chapter has given the basic mechanics of running Xyce in parallel. For general guidance regarding solver options, partitioning options, and other parallel issues, refer to chapter 10. Distributed memory circuit simulation still contains a number of research issues, so obtaining an optimal simulation in parallel is a bit of an art.

Accessing the Microsoft Windows Command Line

Follow the steps below for help with accessing the command line on Windows XP Professional. Consult the operating system documentation for assistance with other versions of Windows.

Figure 2.10. From the Start menu, click Run....
Figure 2.11. Type `cmd` and click OK to open the command line window.

Figure 2.12. The command line window appears and is ready for use.

The following is an illustrated recap of the Command Line Simulation instructions provided earlier.
2.2 Running Xyce

Figure 2.13. Type the full path to the `runxyce` file to execute Xyce. Note that the path must be enclosed in quotation marks if it contains spaces.
Figure 2.14. Using runxyce without any options or a netlist file-name displays a brief help menu.
Figure 2.15. To begin a simulation type the path to runxyce followed by the netlist filename.
Figure 2.16. Output will scroll to the screen. Use `runxyce -h` for assistance with command line options.
3. Simulation Examples with Xyce

Chapter Overview

This chapter provides several simple examples of Xyce usage. An example circuit is provided for each available analysis type.

- Section 3.1 Example Circuit Construction
- Section 3.2 DC Sweep Analysis
- Section 3.3 Transient Analysis
3.1 Example Circuit Construction

This section describes how to use Xyce to create the simple diode clipper circuit shown in Figure 3.1.

While a schematic edit and capture capability is under development, Xyce currently only supports circuit creation via netlist editing. Xyce supports most of the standard netlist entries common to Berkeley SPICE 3F5 and Orcad PSpice. For users who are familiar with PSpice netlists, the differences between PSpice and Xyce netlists are listed in the Xyce Reference Guide [3].

Example: diode clipper circuit

1. Open a new netlist file using a standard text editor (e.g. VI, Emacs, Notepad, etc.).

2. Type the title on the first line of the netlist:

   Diode Clipper Circuit

3. Create a 5V DC voltage source between nodes 1 and 0 by typing the following on a new line:

   VCC 1 0 5V

4. Create another DC voltage source between nodes 3 and 0 by entering the following on a new line:

   VIN 3 0 0V

5. Place the diodes in the circuit between nodes 2 and 1, and nodes 0 and 2, respectively, by entering the following lines:

   D1 2 1 D1N3940
   D2 0 2 D1N3940

6. Enter resistors R1, R2, R3 and R4, respectively:

   R1 2 3 1K
   R2 1 2 3.3K
   R3 2 0 3.3K
   R4 4 0 5.6K

7. Place the capacitor in the circuit:
3.1 Example Circuit Construction

C1 2 4 0.47u

8. Add the diode model to the netlist to complete it as Figure 3.2.

9. Complete the netlist by entering .END on the last line in the file. Save the file as clipper.cir. The complete netlist is shown in Figure 3.2 and the schematic in Figure 3.1.

The netlist in Figure 3.2 illustrates some of the syntax of a netlist input file. Netlists begin with a title (e.g. “Diode Clipper Circuit”), support comments (lines beginning with the “*” character), devices, model definitions and the “.END” statement.

This netlist file is not yet complete and will not run properly using Xyce (see Section 2.2 for instructions on running Xyce) as it lacks an analysis statement. As you proceed in this chapter, you will see how to add the appropriate analysis statement and run the clipper circuit.

Figure 3.1. Schematic of diode clipper circuit with DC and transient voltage sources.
Diode Clipper Circuit
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE: DIODE
* SUBTYPE: RECTIFIER
.MODEL D1N3940 D(
+  IS = 4E-10
+  RS = .105
+  N = 1.48
+  TT = 8E-7
+  CJQ = 1.95E-11
+  VJ = .4
+  M = .38
+  EG = 1.36
+  XTI = -8
+  KF = 0
+  AF = 1
+  FC = .9
+  BV = 600
+  IBV = 1E-4)
*
.END

Figure 3.2. Diode clipper circuit netlist.
3.2 DC Sweep Analysis

In this section an example is given of DC sweep analysis using Xyce. The DC response of the clipper circuit is obtained by sweeping the DC voltage source ($V_{in}$) from -10 to 15 volts in 1 volt steps. For more details about DC analysis see Chapter 7.3 of this manual or the Xyce Reference Guide [3].

Example: DC sweep analysis

To set up and run a DC sweep analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file (clipper.cir) using a standard text editor (e.g. Vi, Emacs, Notepad, etc.).

2. Enter the analysis control statement in the netlist:

   `.DC VIN -10 15 1`

3. Enter the output control statement:

   `.PRINT DC V(3) V(2) V(4)`

4. Save the netlist file and run Xyce on the circuit. For example, to run serial Xyce:

   `> runxyce clipper.cir`

5. Open the results file (clipper.cir.prn) and examine (or plot) the output voltages that were calculated for nodes 3 (Vin), 2 and 4 (Out). Figure 3.4 shows the output plotted as a function of the swept variable Vin.

![Figure 3.4. DC sweep voltages at Vin, node 2 and Vout.](image-url)
Diode Clipper Circuit with DC sweep analysis statement

* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE: DIODE
* SUBTYPE: RECTIFIER
.MODEL D1N3940 D(
  + IS = 4E-10
  + RS = .105
  + N = 1.48
  + TT = 8E-7
  + CJ0 = 1.95E-11
  + VJ = .4
  + M = .38
  + EG = 1.36
  + XTI = -8
  + KF = 0
  + AF = 1
  + FC = .9
  + BV = 600
  + IBV = 1E-4)
*
.END

Figure 3.3. Diode clipper circuit netlist for DC sweep analysis.
3.3 Transient Analysis

This section contains an example of transient analysis in Xyce. In this example the DC clipper circuit of the previous section has been modified so that the input voltage source (Vin) is a time-dependent sinusoidal input source. The frequency of Vin is 1 kHz, and has an amplitude of 10 volts. For more details about transient analysis see Chapter 7.2 of this manual, or see the Xyce Reference Guide [3].

Example: transient analysis

To set up and run a transient analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file file (clipper.cir) using a standard text editor (e.g. VI, Emacs, Notepad, etc.).

2. If you added DC analysis and output statements in the previous example (Figure 3.4), remove them.

3. Enter the analysis control in the netlist:

   \[.TRAN \ 2ns \ 2ms\]

4. Enter the output control statement:

   \[.PRINT \ TRAN \ V(3) \ V(2) \ V(4)\]

5. Modify the input voltage source (Vin) to generate the sinusoidal input signal:

   \[VIN \ 3 \ 0 \ SIN(10V \ 1kHz)\]

6. At this point, the netlist should look similar to the netlist in Figure 3.5. Save the netlist file and run Xyce on the circuit. For example, to run serial Xyce:

   \[> \ runxyce \ clipper.cir\]

7. Open the results file and examine (or plot) the output voltages for nodes 3 (Vin), 2 and 4 (Out). The plot in Figure 3.6 shows the output plotted as a function of time.

The modified netlist is shown in Figure 3.5 and the corresponding results in Figure 3.6.
Diode Clipper Circuit with transient analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE: DIODE
* SUBTYPE: RECTIFIER
.MODEL D1N3940 D(
+ IS = 4E-10
+ RS = .105
+ N = 1.48
+ TT = 8E-7
+ CJ0 = 1.95E-11
+ VJ = .4
+ M = .38
+ EG = 1.36
+ XTI = -8
+ KF = 0
+ AF = 1
+ FC = .9
+ BV = 600
+ IBV = 1E-4)
*
.END

Figure 3.5. Diode clipper circuit netlist for transient analysis.
Figure 3.6. Sinusoidal input signal and clipped outputs.
This page is left intentionally blank
Chapter Overview

This chapter contains introductory material on netlist syntax and usage. Sections include:

- Section 4.1 General Overview
- Section 4.2 Devices Available for Simulation
- Section 4.3 Parameters and Expressions
4.1 General Overview

Introduction

Using a netlist to describe a circuit for Xyce is the primary method for running a circuit simulation. Netlist support within Xyce largely conforms to that used by Berkeley SPICE 3F5 with several new options for controlling functionality unique to Xyce. In a netlist, the circuit is described by a set of element lines which define the circuit elements, their values, the circuit topology (i.e. the connection of the circuit elements), and a variety of control options for the simulation. The first line in the netlist file must be a title and the last line must be ".END". Between these two constraints, the order of the statements is irrelevant.

Netlist Elements

An element line, for which the format is determined by the specific element type, defines each circuit element instance. The general format is given by:

<type><name> <node information> <element information...>

The <type> must be a letter (A through Z) and the <name> follows immediately. For example, RARESISTOR specifies a device of type “R” (for “Resistor”) with a name ARESISTOR. Nodes are separated by spaces, and additional element information required by the device is given after the node list as described in the Netlist Reference section of the Xyce Reference Guide [3]. Xyce ignores character case when reading a netlist such that RARESISTOR is equivalent to rare resistor. The only exception to this case insensitivity occurs when including external files in a netlist where the filename specified in the netlist must have the same case as the actual filename.

A number field may be an integer or a floating-point value. Either one may be followed by one of the following scaling factors:
Node information is given in terms of node names, which are arbitrary character strings. The only requirement is that the ground node is named '0'. There are some restrictions on the circuit topology:

- There can be no loop of voltage sources and/or inductors.
- There can be no cut-set of current sources and/or capacitors.
- Every node must have a DC path to ground.
- Every node must have at least two connections (with the exception of unterminated transmission lines and MOSFET substrate nodes).

The following line provides an example of an element line that defines a resistor between nodes 1 and 3 with a resistance value of 10kΩ.

**Example:**
RARESISTOR 1 3 10K

**Title, Comments and End**

The first line of the netlist is the title line of the netlist. This line is treated as a comment even if it does not begin with an asterisk. It is a common mistake to forget the meaning of this first line and begin the circuit elements on the first line; doing so will probably result in a parsing error.

**Example:**
Test RLC Circuit
The "`.END`" line must be the last line in the netlist.

**Example:** `.END`

Comments are supported in netlists and are indicated by placing an asterisk at the beginning of the comment line. They may occur anywhere in the netlist but they must be at the beginning of a line. Xyce also supports *in-line* comments. An in-line comment is designated by a semicolon and may occur on any line. Everything after the semicolon is taken as a comment and ignored. Any line that begins with leading white space is also considered to be a comment.

**Example:** `* This is a netlist comment.`

**Example:** *WRONG:* `.DC .... * This type of in-line comment is not supported.*`

**Example:** `.DC .... ; This type of in-line comment is supported.*`

### Continuation Lines

Any line that begins with a `+` symbol is a continuation line. Its contents are appended to those of the previous line. If the previous line or lines were comments, the continuation line is appended to the first non-comment line preceding it.

### Netlist Commands

Command elements are used to describe the analysis being defined by the netlist. Examples include analysis types, initial conditions, device models and output control. The Xyce Reference Guide [3] contains a reference for these commands.

**Example:** `.PRINT TRAN V(Vout)`

### Analog Devices

The analog devices supported include most of the standard circuit components normally found in circuit simulators such as SPICE 3F5, PSpice, etc., plus several Sandia specific devices.

**Example:** `D_CR303 N_0065 0 D159700`

To find out more about analog devices see the Xyce Reference Guide [3].
4.2 Devices Available for Simulation

This section describes the different types of analog devices supported in Xyce. These include standard analog devices, sources (dependent and independent) and subcircuits. Each device description has the following information:

- A description and an example of the netlist syntax
- The corresponding model types and descriptions, where applicable
- The corresponding lists of model parameters and descriptions, where applicable
- The associated circuit diagram and model equations, as necessary

These analog devices include all of the standard circuit components needed for most analog circuits. User defined models may also be implemented using the .MODEL (model definition) statement and macromodels as subcircuits using the .SUBCKT (subcircuit) statement.

Analog Devices

Xyce supports many analog devices, including sources, subcircuits and behavioral models. The devices are classified into device types, each of which can have one or more model types. For example, the BJT device type has two model types: NPN and PNP.

The device element statements in the netlist always start with the name of the individual device instance. The first letter of the name determines the device type. The format of the following information depends on the device type and its parameters. The Device Type summary table, Table 4.1, lists all of the analog devices supported by Xyce. Each standard device is then described in more detail in the following sections. Except where noted, the devices are based upon those found in [6].

Table 4.1 is a summary of the analog device types and the form of their netlist formats. For a more complete description of the syntax for supported devices, see the Xyce Reference Guide. [3].

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Designator Letter</th>
<th>Typical Netlist Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitor</td>
<td>C</td>
<td>C&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; [model name] &lt;value&gt; + [IC=&lt;initial value&gt;]</td>
</tr>
<tr>
<td>Inductor</td>
<td>L</td>
<td>L&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; [model name] &lt;value&gt; + [IC=&lt;initial value&gt;]</td>
</tr>
<tr>
<td>Resistor</td>
<td>R</td>
<td>R&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; [model name] &lt;value&gt; + [L=&lt;length&gt;] [W=&lt;width&gt;]</td>
</tr>
</tbody>
</table>
### Netlist Basics

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Designator Letter</th>
<th>Typical Netlist Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diode</td>
<td>D</td>
<td>D&lt;name&gt; &lt;anode node&gt; &lt;cathode node&gt; + &lt;model name&gt; [area value]</td>
</tr>
<tr>
<td>Mutual Inductor</td>
<td>K</td>
<td>K&lt;name&gt; &lt;inductor 1&gt; [&lt;ind. n&gt;*] + &lt;linear coupling or model&gt;</td>
</tr>
<tr>
<td>Independent Voltage Source</td>
<td>V</td>
<td>V&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; [[DC] &lt;value&gt;] + [transient specification]</td>
</tr>
<tr>
<td>Independent Current Source</td>
<td>I</td>
<td>I&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; [[DC] &lt;value&gt;] + [transient specification]</td>
</tr>
<tr>
<td>Voltage Controlled Voltage Source</td>
<td>E</td>
<td>E&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; &lt;+ controlling node&gt; + &lt;+ controlling node&gt; &lt;gain&gt;</td>
</tr>
<tr>
<td>Voltage Controlled Current Source</td>
<td>G</td>
<td>G&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; &lt;+ controlling node&gt; + &lt;+ controlling node&gt; &lt;transconductance&gt;</td>
</tr>
<tr>
<td>Nonlinear Dependent Source (B Source)</td>
<td>B</td>
<td>B&lt;name&gt; &lt;+ node&gt; &lt;- node&gt; + &lt;I or V&gt;=&lt;expression&gt;</td>
</tr>
<tr>
<td>Bipolar Junction Transistor (BJT)</td>
<td>Q</td>
<td>Q&lt;name&gt; &lt;collector node&gt; &lt;base node&gt; &lt;emitter node&gt; [substrate node] &lt;model name&gt; [area value]</td>
</tr>
<tr>
<td>JFET</td>
<td>J</td>
<td>J&lt;name&gt; &lt;drain node&gt; &lt;gate node&gt; &lt;source node&gt; + &lt;model name&gt; + [area value]</td>
</tr>
<tr>
<td>MOSFET</td>
<td>M</td>
<td>M&lt;name&gt; &lt;drain node&gt; &lt;gate node&gt; &lt;source node&gt; + &lt;bulk/substrate node&gt; &lt;model name&gt; + [common model parameter]*</td>
</tr>
<tr>
<td>Transmission Line</td>
<td>T</td>
<td>T&lt;name&gt; &lt;A port + node&gt; &lt;A port - node&gt; + &lt;B port + node&gt; &lt;B port - node&gt; + [ideal specification]</td>
</tr>
<tr>
<td>Voltage Controlled Switch</td>
<td>S</td>
<td>S&lt;name&gt; &lt;+ switch node&gt; &lt;- switch node&gt; + &lt;+ controlling node&gt; &lt;- controlling node + &lt;model name&gt;</td>
</tr>
<tr>
<td>Subcircuit</td>
<td>X</td>
<td>X&lt;name&gt; [node]* &lt;subcircuit name&gt; + [PARAMS: &lt;name&gt;=&lt;value&gt;]*</td>
</tr>
<tr>
<td>PDE Devices</td>
<td>Z</td>
<td>Z&lt;name&gt; &lt;node1&gt; &lt;node2&gt; [node3] + [node4] &lt;model name&gt;</td>
</tr>
</tbody>
</table>

| Table 4.1: Analog Device Quick Reference. |

### 4.3 Parameters and Expressions

In addition to explicit values, the user may use parameters and expressions to symbolize numeric values in the circuit design.
Parameters

A parameter is like a programming variable that represents a numeric value by name. Once you have defined a parameter (declared its name and given it a value) at a particular level in the circuit hierarchy, you can use it to represent circuit values at that level or any level directly beneath it in the circuit hierarchy. One way that you can use parameters is to apply the same value to multiple part instances.

How to Declare and Use Parameters

In order to use a global parameter in a circuit, one must:

- define the parameter using a .PARAM statement within a netlist
- replace an explicit value with the parameter in the circuit

Note that Xyce reserves several keywords that may not be used as parameter names. These are:

- Time
- Vt
- Temp
- GMIN

However, in this release of Xyce, only Time is predefined.

Example: Declaring a parameter

1. Locate the level in the circuit hierarchy at which the .PARAM statement declaring a parameter will be placed. (Note: a global parameter that can be used anywhere in the netlist can be declared by placing the .PARAM statement at the top-most level of the circuit.)

2. Name the parameter and give it a value. The value can be numeric or given by an expression:

   .SUBCKT subckt1 n1 n2 n3
   .PARAM res = 100
   *
   * other netlist statements here
   *
   .ENDS
3. Note: the parameter res can be used anywhere within the subcircuit subckt1 including subcircuits defined within it, but cannot be used outside of subckt1.

Example: Using a parameter in the circuit

1. Find the numeric value that is to be replaced by a parameter: a device instance parameter value, model parameter value, etc. The value being replaced must be accessible with the current hierarchy level.

2. Replace the numeric value with the parameter name contained within braces (\{\}) as in:

   R1 1 2 \{res\}

Expressions

In Xyce, an expression is a mathematical relationship that may be used any place one would use a number (numeric or boolean). Except in the case of expressions used in analog behavioral modeling sources (see Chapter 6) Xyce evaluates the expression to a value when it reads in the circuit netlist, not each time its value is needed. It is therefore necessary that all terms in an expression be known at the beginning of the run.

To use an expression in a circuit netlist:

1. Locate the value to be replaced (component, model parameter, etc.).

2. Substitute the value with an expression utilizing the \{\} syntax:

   \{expression\}

   where expression can contain any of the following:

   - available operators from those in Table 4.2
   - included functions from those in Table 4.3
   - user-defined functions
   - user-defined parameters that are within scope
   - literal operands

The braces (\{\}) instruct Xyce to evaluate the expression and use the resulting value. Additional time-dependent constructs are available in expressions used in analog behavioral modeling sources (see Chapter 6).
Example: Using an expression

Scaling the DC voltage of a 12V independent voltage source, designated $V_F$, by some factor can be accomplished by the following netlist statements (in this example the factor is 1.5):

```
.PARAM FACTORV=1.5
VF 3 4 {FACTORV*12}
```

**Xyce** will evaluate the expression to $12 \times 1.5$ or 18 volts.

<table>
<thead>
<tr>
<th>Class of operator</th>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>+</td>
<td>addition or string concatenation</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td></td>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>logical</td>
<td>~</td>
<td>unary NOT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>boolean OR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>boolean XOR</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>boolean AND</td>
</tr>
<tr>
<td>relational</td>
<td>==</td>
<td>equality</td>
</tr>
<tr>
<td></td>
<td>!=</td>
<td>non-equality</td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
<td>greater-than</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>greater-than or equal</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>less-than</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>less-than or equal</td>
</tr>
</tbody>
</table>

**Table 4.2. Expression operators**

---

1 Logical and relational operators are used only with the IF() function.
<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(x)</td>
<td></td>
<td>absolute value of x</td>
</tr>
<tr>
<td>SQRT(x)</td>
<td>√x</td>
<td>square root of x</td>
</tr>
<tr>
<td>MIN(x,y)</td>
<td>min(x, y)</td>
<td>minimum of x and y</td>
</tr>
<tr>
<td>MAX(x,y)</td>
<td>max(x, y)</td>
<td>maximum of x and y</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>$e^x$</td>
<td>$e$ to the x power</td>
</tr>
<tr>
<td>LN(x)</td>
<td>ln(x)</td>
<td>log base $e$</td>
</tr>
<tr>
<td>LOG(x)</td>
<td>log(x)</td>
<td>log base 10</td>
</tr>
<tr>
<td>SIN(x)</td>
<td>sin(x)</td>
<td>x in radians</td>
</tr>
<tr>
<td>ASIN(x)</td>
<td>arcsin(x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>SINH(x)</td>
<td>sinh(x)</td>
<td>x in radians</td>
</tr>
<tr>
<td>ASINH(x)</td>
<td>sinh$^{-1}$(x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>COS(x)</td>
<td>cos(x)</td>
<td>x in radians</td>
</tr>
<tr>
<td>ACOS(x)</td>
<td>arccos(x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>COSH(x)</td>
<td>cosh(x)</td>
<td>x in radians</td>
</tr>
<tr>
<td>ACOSH(x)</td>
<td>cosh$^{-1}$(x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>TAN(x)</td>
<td>tan(x)</td>
<td>x in radians</td>
</tr>
<tr>
<td>ATAN(x)</td>
<td>arctan(x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>TANH(x)</td>
<td>tanh(x)</td>
<td>x in radians</td>
</tr>
<tr>
<td>ATANH(x)</td>
<td>tanh$^{-1}$(x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>ATAN2(x,y)</td>
<td>arctan(y/x)</td>
<td>result in radians</td>
</tr>
<tr>
<td>SGN(x)</td>
<td>+1 if x &gt; 0; 0 if x = 0; -1 if x &lt; 0</td>
<td>sign value of x</td>
</tr>
<tr>
<td>STP(x)</td>
<td>1 if x &gt; 0; 0 otherwise</td>
<td>suppress a value until a given time</td>
</tr>
<tr>
<td>URAMP(x)</td>
<td>x if x &gt; 0; 0 otherwise</td>
<td>ramp function</td>
</tr>
<tr>
<td>IF(t,x,y)</td>
<td>x if t is true, y otherwise</td>
<td>$t$ is an expression using the relational operators in Table 4.2</td>
</tr>
<tr>
<td>DDT(x)</td>
<td>$\frac{dx}{dt}$</td>
<td>time derivative of x</td>
</tr>
<tr>
<td>SDT(x)</td>
<td>$\int x(t)dt$</td>
<td>time integral of x</td>
</tr>
</tbody>
</table>
5. Working with .MODEL Statements and Subcircuit Models

Chapter Overview

This chapter contains model ideas and a summary of the ways to create and modify models. Sections include:

- Section 5.1 Definition of a Model
- Section 5.2 Model Organization
5.1 Definition of a Model

A model describes the electrical performance of a part, such as a specific vendor’s version of a 2N2222 transistor. To simulate a part requires specification of simulation properties. These properties define the model of the part.

Depending on the given device type and the requirements of the circuit design, a model is specified using a model parameter set, a subcircuit netlist, or both.

In general, model parameter sets define the parameters used in ideal models of specific device types, and subcircuit netlists allow the user to combine ideal device models to simulate more complex effects. For example, one could simulate a bipolar transistor using the Xyce BJT device by specifying model parameters extracted to fit the simulation behavior to the behavior of the part used, or one could develop a subcircuit macro-model of a capacitor that adds effects like lead inductance and resistance to the basic capacitor device.

Both methods of defining a model use a netlist format, with precise syntax rules as described below.

Defining models using model parameters

**Xyce** currently has no built-in part models. However, models can be defined for a device by changing some or all of the model parameters from their defaults via the .MODEL statement. For example:

\[
\text{M5 3 2 1 0 MLOAD1} \\
\text{.MODEL MLOAD1 NMOS (LEVEL=3 VTO=0.5 CJ=0.025pF)}
\]

This example defines a MOSFET device M5 that is an instance of a part described by the model parameter set MLOAD1. The MLOAD1 parameter set is defined in the .MODEL statement.

Most device types in **Xyce** support some form of model parameters. Consult the **Xyce** Reference Guide [3] for the model parameters supported by each device type.

Defining models using subcircuit netlists

In **Xyce**, models may also be defined using the subcircuit syntax: .SUBCKT/.ENDS. This syntax includes:

- **netlists** to define the configuration and function of the part.
5.1 Definition of a Model

- variable input parameters to modify the model.

See Figure 5.1 for an example.

```plaintext
****other devices
X5 5 6 7 8 l3dsc1 PARAMS: ScaleFac=2.0
X6 9 10 11 12 l3dsc1
****more netlist commands

*** SUBCIRCUIT: l3dsc1
*** Parasitic Model: microstrip
*** Only one segment
.SUBCKT l3dsc1 1 3 2 4 PARAMS: ScaleFac=1.0
C01 1 0 4.540e-12
RG01 1 0 7.816e+03
L1 1 5 3.718e-08
R1 5 2 4.300e-01
C1 2 0 4.540e-12
RG1 2 0 7.816e+03
C02 3 0 4.540e-12
RG02 3 0 7.816e+03
L2 3 6 3.668e-08
R2 6 4 4.184e-01
C2 4 0 4.540e-12
RG2 4 0 7.816e+03
CM012 1 3 5.288e-13
KM12 L1 L2 2.229e-01
CM12 2 4 {5.288e-13*ScaleFac}
.ENDS
.ENDS
```

**Figure 5.1.** Example subcircuit model.

In this example, a subcircuit model called l3dsc1 implementing one part of a microstrip transmission line is defined between the .SUBCKT/.ENDS lines, and two different instances of the subcircuit are used in the X lines. This somewhat artificial example shows how input parameters are used; the last capacitor in the subcircuit is scaled by the input parameter ScaleFac. If input parameters are not specified on the X line (as in the case of device X6), then the default values specified on the .SUBCKT line are used. Non-default values are specified on the X line using the PARAMS: keyword. For precise syntax consult the Xyce Reference Guide [3].
Subcircuit Hierarchy

**Xyce** supports the definition of subcircuits within other subcircuits. Each subcircuit definition introduces a new level in the circuit hierarchy with the top level being the main circuit. If a second level is defined, it is composed of the subcircuits in the main circuit and each subsequent level is composed of the subcircuits contained in the previous level. A subcircuit may also contain other definitions such as models via the **.MODEL** statement, parameters via the **.PARAM** statement, and functions via the **.FUNC** statement.

In this context, the subcircuit defines the “scope” for the definitions it contains. That is, *the definitions contained within a subcircuit can be used within that subcircuit and/or within any subcircuit it contains*. Any definitions occurring in the main circuit have global scope and can be used anywhere in the circuit. A name, such as a model, parameter, function or subcircuit name, occurring in a definition at one level of a circuit hierarchy can be redefined at any lower level contained directly by the subcircuit. In this case, the new definition applies at the given level and those below.

In the following example, the model named **MOD1** can be used in subcircuits **SUB1** and **SUB2** but not in the subcircuit **SUB3**. The parameter **P1** has a value of **10** in subcircuit **SUB1** and a value of **20** in subcircuit **SUB2**.

```
.SUBCKT SUB1 1 2 3 4
.MODEL MOD1 NMOS(LEVEL=2)
.PARAM P1=10
*
* subcircuit devices omitted for brevity
*
.SUBCKT SUB2 1 3 2 4
.PARAM P1=20
*
* subcircuit devices omitted for brevity
*
.ENDS
.ENDS

.SUBCKT SUB3 1 2 3 4
*
* subcircuit devices omitted for brevity
*
.ENDS
```

**Figure 5.2.** Example subcircuit model.
5.2 Model Organization

While it is always possible to make a self-contained netlist in which all models for all parts are included along with the circuit definition, it is often more convenient to organize frequently-used models into separate model libraries. Xyce provides a very simple mechanism that allows this organization. Models are simply collected into model library files, and then accessed by netlists as needed by insertion of an .INCLUDE directive. This section describes the process in detail.

Model libraries

Device model and subcircuit definitions may be organized into model libraries. These libraries are text files (similar to netlist files) that have one or more model definitions. Many users choose to give model library files names that end with .lib, but the file may be named using any convention the user chooses.

In general, most users create model libraries files that typically include similar model types. In these files, the header comments describe the models therein.

Model library configuration

In Xyce, model libraries are used by inserting a .INCLUDE statement into a netlist. Once a file is included, its contents are available to the netlist just as if the entire contents had been inserted directly into the netlist.

As an example, one might create the following model library file called bjtmodels.lib, containing .MODEL statements for common types of bipolar junction transistors:

*bjtmodels.lib
*Bipolar transistor models
.MODEL Q2N2222 NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=5 Ne=1.307
+ Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+ Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+ Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)

.MODEL 2N3700 NPN (IS=17.2E-15 BF=100)

.MODEL 2N2907A PNP (IS=1.E-12 BF=100)

The models Q2N2222, 2N3700 and 2N2907A could then be used in a netlist by including the bjtmodels.lib file.
Because the contents of an included file are simply inserted into the netlist at the point where the .INCLUDE statement appears, the scoping rules for .INCLUDE statements is the same as for other types of definitions as outlined in the preceding section. Note that the path to the library file is assumed to be relative to the execution directory, but absolute pathnames are permissible. Note also that the entire file name, including its “extension” must be specified. There is no assumed default extension.
6. Analog Behavioral Modeling

Chapter Overview

This chapter contains a description of analog behavioral modeling in Xyce. Sections include:

- Section 6.1 Overview of Analog Behavioral Modeling
- Section 6.2 Specifying ABM Devices
- Section 6.3 Guidance for ABM Use
6.1 Overview of Analog Behavioral Modeling

The analog behavioral modeling capability of Xyce provides for flexible descriptions of electronic components in terms of a transfer function or lookup table. In other words, a mathematical relationship is used to model a circuit segment removing the need for component by component design.

The primary device used for analog behavioral modeling in Xyce is the B device, or non-linear dependent source. A B device can serve as a voltage or current source, and by using expressions dependent on voltages and currents elsewhere in the circuit the user can produce any desired behavior.

6.2 Specifying ABM Devices

ABM devices (B devices) are specified in a netlist the same way as other devices. Customizing the operational behavior of the device is achieved by defining an ABM expression describing how inputs are transformed into outputs.

For example, the pair of lines below would provide exactly the same behavior as a 10K resistor between nodes 1 and 2. It is written to be a current source with the current specified using Ohm’s law and the constant resistance.

```
.PARAM Res1=10K
Blinearres 1 2 I={(V(2)-V(1))/Res1}
```

A nonlinear resistor could be specified similarly:

```
.PARAM R1=0.15
.PARAM R2=6
.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}

.FUNC Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

Bnlr 4 2 V = {I(Vmon) * IF(+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
```
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2 + )
+ )
}

In this example, Bnlr provides a voltage between nodes 4 and 2, and the voltage is determined using Ohm's law with a resistance that is a function of the voltage on node 101 and a number of parameters. These two examples demonstrate how the B source can be used either as a voltage source (by specifying V={expression}) or as a current source (with I={expression}).

Note that unlike expressions used in parameters or function declarations, expressions in the nonlinear dependent source may contain voltages and currents from other parts of the circuit, or even explicit time-dependent functions. These expressions are evaluated every time the current or voltage through the source are needed.

### Additional constructs for use in ABM expressions

ABM expressions follow the same rules as other expressions in a netlist with the additional ability to specify signals (node voltages and voltage source currents) and explicitly time-dependent functions in the expression. In ABM expressions, refer to signals by name. Xyce recognizes the following constructs in ABM expressions:

- V(<node name>)
- V(<node name>,<node name>)
- I(<voltage source name>)
- The variable TIME
- Lookup tables

In a hierarchical circuit (a circuit with possibly nested levels of subcircuits), voltage source names that appear in an ABM expression must be the name of a voltage source in the same subcircuit as the ABM device. Similarly, node names in an ABM expression must be the node names of one or more devices in the same subcircuit as the ABM device.

### Lookup Tables in Analog Behavioral Modeling

Lookup tables provide a means of specifying a piecewise-linear function in an expression. A table expression is specified with the keyword TABLE followed by an expression that is evaluated as the independent variable of the piecewise linear function, followed by a list of pairs of independent variable/dependent variable values. For example
Example: \[ B1 \ 0 \ V = \{ \text{TABLE} \ \{ \text{time} \} = (0, 0) \ (1, 2) \ (2, 4) \ (3, 6) \} \]

will produce a voltage source whose voltage is a simple linear function of time. At \( t = 0 \) the voltage is 0 volts, at time \( t = 1 \) s the voltage is 2 volts, and at times in between the voltage is determined by linear interpolation.

The independent variable of the table source does not have to be a simple expression:

Example: \[ B1 \ 0 \ V = \{ \text{TABLE} \ \{ V(5) - V(3)/4 + I(V6)*\text{Res1} \} = (0, 0) \ (1, 2) \ (2, 4) \ (3, 6) \} \]

Alternate behavioral modeling sources

In addition to the primary nonlinear dependent source, the \( B \) source, \textit{Xyce} also supports the PSpice extensions to the standard Spice voltage- and current-controlled sources, the \( E \), \( F \), \( G \) and \( H \) sources. These sources are provided for PSpice compatibility, and are converted internally into \( B \) sources. See the Netlist Reference chapter of the \textit{Xyce} Reference Guide \[3\] for the syntax of these compatibility devices.

6.3 Guidance for ABM Use

Users frequently use ABM devices to provide output post-processing. For example, if a user was interested in the absolute value, or the log of an output voltage, that user might create a ABM circuit element that calculated the desired output value.

Using ABM sources in this manner is a bad practice. By creating a circuit element whose only purpose is post-processing, \textit{Xyce} is forced to include this bogus device in the circuit, and the corresponding nonlinear solve. Often, this causes unnecessary solver problems. If post-processing is the goal, it is much better to use expressions directly on the \texttt{.PRINT} line. This is a supported capability as of \textit{Xyce} Release 2.1.

* Bad example
  \[ B1 \ \text{test1} \ 0 \ V = (\text{abs}(I(VMON))) \times 1.0 \times 10 \]
  \[ \text{VIN} \ 1 \ 0 \ \text{DC} \ 5V \]
  \[ \text{R1} \ 1 \ 2 \ 2K \]
  \[ \text{D1} \ 3 \ 0 \ \text{DMOD} \]
  \[ \text{VMON} \ 2 \ 3 \ 0 \]
  \[ .\text{MODEL} \ \text{DMOD} \ D \ \{ \text{IS} = 100 FA \} \]
  \[ .\text{DC} \ \text{VIN} \ 5 \ 5 \ 1 \]
  \[ .\text{PRINT} \ \text{DC} \ I(VMON) \ V(3) \ V(\text{test1}) \]

An example of a “bad use” of ABM sources can be found the above code fragment. The source \( B1 \), is only in the circuit to provide a post-processing output. It doesn't play a
functional role in the circuit, but \textbf{Xyce} would still be forced to include it in the problem it is attempting to solve. A much better solution is to get rid of B1, and replace it with an expression in the \texttt{.PRINT} line. A better solution to the above problem is given here:

\begin{verbatim}
* Good example
VIN 1 0 DC 5V
R1 1 2 2K
D1 3 0 DMOD
VMON 2 3 0
.MODEL DMOD D (IS=100FA)
.DC VIN 5 5 1
.PRINT DC I(VMON) V(3) {abs(I(VMON))*1.0e-10}
\end{verbatim}

For a more detailed explanation of how to use expressions in the \texttt{.PRINT} line, see section[9.1] or the \textbf{Xyce} Reference Guide [3].
Chapter Overview

This chapter contains a description of the different analysis types available in Xyce. It includes the following sections:

- Section 7.1 Introduction
- Section 7.2 Transient Analysis
- Section 7.3 DC Analysis
- Section 7.4 STEP Parametric Analysis
7.1 Introduction

Several simulation analysis options are supported within Xyce. For basic analysis, Xyce currently supports DC and transient analysis; AC analysis is intended to be supported in a future release. STEP parametric analysis, which applies an outer parameter loop to either DC or transient analysis is also available.

7.2 Transient Analysis

The transient response analysis simulates the response of the circuit from \texttt{TIME=0} to a specified time. Throughout a transient analysis, any or all of the independent sources may have time-dependent values.

In Xyce (and most other circuit simulators), the transient analysis begins by performing its own bias point calculation at the beginning of the run, using the same method as used for DC sweep. This is required to set the initial conditions for the transient solution as the initial values of the sources may differ from their DC values.

.TRAN Statement

To run a transient simulation, the circuit netlist file must contain a \texttt{.TRAN} command.

Example:

\begin{verbatim}
.TRAN 100us 300ms
.TRAN 100p 12.05u 9.95u
\end{verbatim}

For a detailed explanation of the \texttt{.TRAN} statement, see the Xyce Reference Guide \cite{3}). In addition to a \texttt{.TRAN} statement, the netlist must contain one of the following:

- an independent, transient source (see Table \ref{table:7.1}),
- an initial condition on a reactive element, or
- a time-dependent behavioral modeling source (see Chapter \ref{chapter:6})

Defining a Time-Dependent (transient) Source

Overview of Source Elements

Source elements, either voltage or current, are entered in the netlist file as described in the Xyce Reference Guide \cite{3}. Table \ref{table:7.1} list the time-dependent sources available in Xyce for
either voltage or current. For voltage sources, the name is preceded by the letter \( V \) while current sources are preceded by the letter \( I \).

<table>
<thead>
<tr>
<th>Source Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP</td>
<td>Exponential Waveform</td>
</tr>
<tr>
<td>PULSE</td>
<td>Pulse Waveform</td>
</tr>
<tr>
<td>PWL</td>
<td>Piecewise Linear Waveform</td>
</tr>
<tr>
<td>SFFM</td>
<td>Frequency-modulated Waveform</td>
</tr>
<tr>
<td>SIN</td>
<td>Sinusoidal Waveform</td>
</tr>
</tbody>
</table>

Table 7.1. Summary of time-dependent sources supported by Xyce.

To use one of these time-dependent or transient sources, the user must place the source element line in the netlist and characterize the transient behavior using the appropriate parameters. Each transient source element has a separate set of parameters dependent on its transient behavior. In this way, the user can create analog sources which produce sine wave, square pulse, exponential pulse, single-frequency FM, and piecewise linear waveforms.

Defining Transient Sources

To define a transient source:

- Select one of the supported sources: independent voltage or current source.
- Choose a transient source type from Table 7.1.
- Provide the transient parameters (see the Xyce Reference Guide [3]) to fully define the source.

Below is an example of an independent sinusoidal voltage source in a circuit netlist. It creates a voltage source between nodes 1 and 5 that oscillates sinusoidally between -5V and +5V with a frequency of 50 KHz.

Example: \[ V_{example} 1 \ 5 \ SIN(-5V \ 5V \ 50KHz) \]

Transient Calculation Time Steps

During the simulation, Xyce uses a calculation time step that is continuously adjusted for accuracy and efficiency (see [7] and [8]). During periods of circuit idleness the calculation
time step is increased, and during dynamic portions of the waveform it is decreased. The maximum internal step size can be controlled by specifying the step ceiling value in the .TRAN command (see the Xyce Reference Guide [3]).

The internal calculation time steps used might not be consistent with the output time steps requested by the user. By default Xyce outputs solution results at every time step it calculates. If the user selects output timesteps via the .OPTIONS OUTPUT statement (see Chapter 9) then Xyce will output results for the closest time step that follows the time requested by the user. There is currently no mechanism for forcing Xyce to output at precise user-specified times.

**Checkpointing and Restarting**

The .OPTIONS RESTART command (in the netlist) is used to control all checkpoint output and restarting. Checkpointing and associated restart can be extremely useful for long simulations. In essence, Xyce allows the user to save the state of the simulation during a run (at intervals the user specifies) (checkpointing). This checkpoint data can then be read in to restart the simulation from any of the saved (checkpointed) time points.

**Checkpointing Command Format**

- .OPTIONS RESTART PACK=<0|1> JOB=<job name> [INITIAL_INTERVAL=<interval> [t0] <i0> [t1] <i1>...]]]  
  PACK=<0|1> indicates whether the restart data files will contain byte packed data(1) or not(0). JOB=<job name> identifies the prefix for restart files. The actual restart files will be the job name appended with the current simulation time (e.g. name1e-05 for JOB=name and simulation time 1e-05 seconds). Furthermore, the INITIAL_INTERVAL=<interval> identifies the initial interval time used for restart output. The <tx ix> intervals identify times (tx) at which the output interval (ix) will change. This functionality is identical that described for the .OPTIONS OUTPUT command (see Section 9.1).

- Example - generate checkpoints at every time step (default):
  
  .OPTIONS RESTART JOB=checkpt

- Example - generate checkpoints every 0.1 µs:
  
  .OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us

- Example - generate unpacked checkpoints every 0.1 µs:
  
  .OPTIONS RESTART PACK=0 JOB=checkpt INITIAL_INTERVAL=0.1us

- Example - Initial interval of 0.1 µs, at 1 µs in the simulation, change to interval of 0.5 µs, and at 10 µs change to an interval of 0.1 µs:
7.3 DC Analysis

The DC sweep analysis capability in Xyce carries out a sweep, in DC mode, on a circuit. DC sweep is supported for a source (current or voltage), through a range of specified values. As the sweep proceeds, the bias point is computed for each value in the specified range of the sweep.

If the variable to be swept is a voltage or current source, a DC source must be used. The DC value is set in the netlist (see the Xyce Reference Guide [3]). In simulating the DC response of an analog circuit, Xyce eliminates any time dependence from the circuit. This is accomplished by treating all capacitor elements as open circuits, all inductor elements as short circuits and using only the DC values of both voltage and current sources.
.DC Statement

One specifies a .DC analysis with a .DC line in the netlist. Some examples of typical .DC lines are:

**Example:**
```
.DC M1:L 7u 5u -1u
.DC OCT V0 0.125 64 2
.DC DEC R1 100 10000 3
.DC TEMP LIST 10.0 15.0 18.0 27.0 33.0
```

The examples include all four types of sweep - linear, octave, decade, and list. For a complete description of each of these, see the Xyce Reference Guide [3].

Setting Up and Running a DC Sweep

Following the example given in Section 3.2, the diode clipper circuit netlist is shown in Figure 7.1 with a DC sweep analysis specified. Here, the voltage source \( V_{in} \) is swept from -10 to 15 in 1 volt increments, resulting in 26 DC operating point calculations. Note also that the default setting for \( V_{in} \) is ignored during these calculations. All other source values use the specified values (\( V_{CC} = 5V \) in this case).

Running Xyce on this netlist produces an output results file named clipper.cir.prn. Obtaining this file requires that the .PRINT DC line be specified. Plotting this data produces the graph shown in Figure 7.2.

OP Analysis

Xyce also supports .OP analysis statements. In Xyce, .OP should be considered as a shorthand for a single step DC sweep, in which all the default operating point values are used. One can also consider .OP analysis to be the operating point calculation which would occur as the initial step to a transient calculation, without the subsequent time steps.

This capability was mainly added so that the code would be able to handle legacy netlists which used this type of analysis statement. In most versions of SPICE, using .OP will result in extra output which is not available from a DC sweep. That additional output capability has not yet been implemented in Xyce.
Diode Clipper Circuit
** Voltage Sources
VCC 1 0 5V VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940 D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
** GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE: DIODE
* SUBTYPE: RECTIFIER
.MODEL D1N3940 D(
+     IS = 4E-10
+     RS = .105
+     N = 1.48
+     TT = 8E-7
+     CJ0 = 1.95E-11
+     VJ = .4
+     M = .38
+     EG = 1.36
+     XTI = -8
+     KF = 0
+     AF = 1
+     FC = .9
+     BV = 600
+     IBV = 1E-4)
* .END

**Figure 7.1.** Diode clipper circuit netlist for DC sweep analysis.
Figure 7.2. DC sweep voltages at Vin, node 2 and Vout.
7.4 STEP Parametric Analysis

The .STEP command performs a parametric sweep for all the analyses of the circuit. When this command is invoked, all of the typical analysis, such as .DC or .TRAN analysis are performed at each parameter step.

This capability is very similar to the STEP capability in PSPICE [4], but not identical. In Xyce, .STEP can be used to sweep over any device instance or device model parameter, as well as the circuit temperature. Currently, there is not a capability for sweeping global parameters, as specified by a .PARAM statement (Section 4.3).

.STEP Statement

One specifies a .STEP analysis by simply adding a .STEP line to a netlist. Note that unlike .DC, .STEP by itself is not an adequate analysis specification, as it merely specifies an outer loop around the normal analysis. There needs still needs to be a standard analysis line, either specifying .TRAN or .DC analysis.

Some examples of typical .STEP lines are:

Example:

```
.STEP M1:L 7u 5u -1u
.STEP OCT V0 0.125 64 2
.STEP DEC R1 100 10000 3
.STEP TEMP LIST 10.0 15.0 18.0 27.0 33.0
```

.STEP has a format similar to that of the .DC specification. In the first example, M1:L is the name of the parameter, 7u is the initial value of the parameter, 5u is the final value of the parameter, and -1u is the step size. Like .DC, .STEP in Xyce can also handle sweeps by decade, octave or specified lists of values. For a complete explanation of each type of sweep, consult the Xyce Reference Guide [3].

Sweeping over a Device Instance Parameter

The first example uses M1:L as the parameter, but it could have used any model or instance parameter that existed in the circuit. Internally, Xyce handles the parameters for all device models and device instances in the same way. You can uniquely identify any parameter by specifying the device instance name, followed by a colon (:), followed by the specific parameter name. For example, all the MOSFET models have an instance parameter for the channel length, L. If you have a MOSFET instance specified in a netlist, named M1, then the full name for M1’s channel length parameter is M1:L.

A simple application of .STEP to a device instance is given in figure 7.3. This is the same
diode clipper circuit as was used in the transient analysis chapter, except that a single line
(in red font) has been added. The .STEP line will cause Xyce to sweep the resistance of
the resistor, R4, from 3.0 KOhms to 15.0 KOhms, in increments of 2.0 KOhms. This means
that a total of seven transient simulations will be performed, each one with a different value
for R4.

As the circuit is executed multiple times, the file resulting output is a little more sophisti-
cated. The .PRINT statement is still used in much the same way as before. However, .prn
output file contains the concatenated output of each .STEP increment. For details of how
.STEP changes output files, see the end of this section.

Sweeping over a Device Model Parameter

Sweeping a model parameter can be done in an identical manner to an instance parameter.
Figure 7.4 contains the same circuit as in figure 7.3 but with a new .STEP line added. The
new .STEP line refers to a model parameter, D1N3940:IS. Note that separate .STEP lines is
the correct way to specify multiple parameters for .STEP analysis. Each parameter needs
its own separate line. In this respect, the .STEP line syntax differs from the .DC line syntax.

Sweeping over Temperature

It is also possible to sweep over temperature. To do so, simply specify temp as the param-
eter name. It will work in the same manner as .STEP when applied to model and instance
parameters.

Special cases: Sweeping Independent Sources, Resistors,
Capacitors

For some devices, there is generally only one parameter that one would want to actually
sweep. For example, a linear resistor’s only parameter of interest is the resistance, R. Sim-
ilarly, for a DC voltage or current source, one is usually only interested in the magnitude of
the source. Finally, linear capacitors generally only have the capacitance, C, as a param-
ter of interest. To make things easier for the user, these three types of devices have default
parameters. Examples of usage are given below.

Example:

.STEP R4 3.0K 15.0K 2.0K
.STEP VCC 4.0 6.0 1.0
.STEP ICC 4.0 6.0 1.0
.STEP C1 0.45u 0.50u 0.1u

Independent sources require some extra explanation. There are a number of different
Transient Diode Clipper Circuit with step analysis

* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)

* Analysis Command
.TRAN 2ns 2ms

* Output
.PRINT TRAN V(3) V(2) V(4)

* Step statement
.STEP R4:R 3.0K 15.0K 2.0K

* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940

* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K

* Capacitor
C1 2 4 0.47u

* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE: DIODE SUBTYPE: RECTIFIER
.MODEL D1N3940 D(
+ IS = 4E-10
+ RS = .105
+ N = 1.48
+ TT = 8E-7
+ CJO = 1.95E-11
+ VJ = .4
+ M = .38
+ EG = 1.36
+ XTI = -8
+ KF = 0
+ AF = 1
+ FC = .9
+ BV = 600
+ IBV = 1E-4)

* .END

Figure 7.3. Diode clipper circuit netlist for step transient analysis.
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
* Step statements
.STEP R4:R 3.0K 15.0K 2.0K
.STEP D1N3940:IS 2.0e-10 6.0e-10 2.0e-10
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE: DIODE SUBTYPE: RECTIFIER
.MODEL D1N3940 D(
+   IS = 4E-10
+   RS = .105
+   N = 1.48
+   TT = 8E-7
+   CJO = 1.95E-11
+   VJ = .4
+   M = .38
+   EG = 1.36
+   XTI = -8
+   KF = 0
+   AF = 1
+   FC = .9
+   BV = 600
+   IBV = 1E-4)
*
.END

Figure 7.4. Diode clipper circuit netlist for 2-step transient analysis.
types of independent source, and only some of them have default parameters. Sources which are subject to .DC sweeps (swept sources) do not have a default parameter, as this could easily lead to infinite loops. The various independent source defaults are defined in the table.

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sinusoidal source</td>
<td>V0 (DC value, Offset)</td>
</tr>
<tr>
<td>Exponential source</td>
<td>V1 (DC value, Initial value)</td>
</tr>
<tr>
<td>Pulsed source</td>
<td>V2 (Pulsed value)</td>
</tr>
<tr>
<td>Constant, or DC source</td>
<td>V0 (Constant value)</td>
</tr>
<tr>
<td>Piecewise Linear source</td>
<td>No default</td>
</tr>
<tr>
<td>SFFM source</td>
<td>No default</td>
</tr>
<tr>
<td>Swept source (specified on a .DC line)</td>
<td>No default</td>
</tr>
</tbody>
</table>

Table 7.2: Default parameters for independent sources.

Output files

A .STEP simulation can be thought of as several distinct executions of the same circuit netlist. The output data, as specified by a .PRINT line, however, goes to a single (*.prn) file. For convenience, a second auxiliary file is also created, with the *.res suffix.

The example file given in figure 7.3 has a filename of clip.cir. When run, it will produce files clip.cir.res and clip.cir.prn. clip.cir.res contains one line for each step, showing what parameter value was used on that step. clip.cir.prn is the familiar output format, but the INDEX field recycles to zero each time a new step begins. Since the default behavior distinguishes each step’s output only by recycling the INDEX field to zero, it can be beneficial to add the sweep parameters to the .PRINT line. For the default file format (format=std), these sweep parameters will not be included automatically, so for plotting purposes it is usually best to specify them by hand.

If using the default .prn file format (format=std), the output file resulting from a .STEP simulation will just be a simple concatenation. If using format=probe, the data for each execution of the circuit will be in distinct sections of the file, and it should be easy to plot the results, using PROBE. If using format=tecplot, the results of each .STEP simulation will be in a distinct tecplot zone.
This page is left intentionally blank
Chapter Overview

This chapter includes the following sections:

- Section 8.1 *Homotopy Algorithms Overview*
- Section 8.2 *Examples*
8.1 Homotopy Algorithms Overview

The most difficult type of numerical nonlinear circuit problem to solve is a DC operating point. DC operating point analysis cannot rely on a good initial guess whereas a transient analysis can use the results of the previous time step. Additionally, operating points often have multiple solutions.

Homotopy methods can often provide solutions to difficult nonlinear problems when more conventional numerical methods fail. In recent years, these techniques have been applied to circuit analysis. As of the Xyce Version 2.1 release, some of these algorithms have been added to Xyce. This chapter gives an introduction to the usage of homotopy algorithms (also called continuation algorithms) in Xyce. For a more complete description of solver options, see the Xyce Reference Guide [3].

HOMOTOPY Algorithms Available in Xyce

There are two general types of homotopy which are available in Xyce. The first (which is set with \texttt{.options nonlin continuation=1}), is a simple natural parameter homotopy, in which the homotopy parameter is an already-defined input parameter to a device model or instance. This algorithm can be useful, but often is not. The most obvious natural parameters to use (the magnitudes of independent sources) tend to lead to turning points in the continuation.

The second is an algorithm which is designed especially for MOSFET circuits [9]. This algorithm involves two internal MOSFET model parameters, one for the MOSFET gain, and the other for the nonlinearity of the current-voltage relationship. This algorithm is invoked with \texttt{.options nonlin continuation=2}. This algorithm has proved to be very effective in large MOSFET circuits.

8.2 Examples

MOSFET Homotopy

Figure 8.1 contains a MOSFET homotopy example netlist. Note that this is a usage example - the circuit itself does not require homotopy to run. Circuits which are complex enough to require homotopy would not fit on a single page. The lines pertinent to the homotopy algorithm are highlighted in red.

Explanation of Parameters, Best Practice

Note that this example shows one set of options, but there are a number of other combinations of options that will work.
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3

* HOMOTOPY Options
.options device voltlim=0
.options nonlin continuation=2
.options loca stepper=0 predictor=0 stepcontrol=1
+ initialvalue=0.0 minvalue=-1.0 maxvalue=1.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200

VDDdev VDD 0 5V
RIN IN 1 1K
VIN1 1 0 5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1 VOUT 0 10K
C2 VOUT 0 0.1p
MN1 VOUT IN 0 0 CD4012_NMOS L=5u W=175u
MP1 VOUT IN VDD VDD CD4012_PMOS L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END

Figure 8.1. Example MOSFET homotopy netlist.
There are a number of "best practice" rules, which are illustrated by the example in figure 8.1. They are:

- **voltlim=0.** This is absolutely required - the homotopy algorithms will usually break if used in conjunction with voltage limiting.

- **continuation=2.** This specifies that we are using the special MOSFET homotopy. This is a 2-pass homotopy, in which first a parameter having to do with the gain is swept from 0 to 1, and then a parameter relating to the nonlinearity of the transfer curve is swept from 0 to 1.

- **initialvalue=0.0.** This is required.

- **maxvalue=1.0.** This is required.

- **stepcontrol=1.** This specifies that the homotopy steps are adaptive, rather than constant. This is recommended.

- **maxsteps=100.** This sets the maximum number of continuation steps for each parameter. For the special MOSFET continuation (which has 2 parameters), this means a maximum of 200 steps.

- **maxnliters=200.** This is the maximum number of nonlinear iterations, and has precedence over the similar number which can be set on the .options nonlin line.

- **aggressiveness=1.0.** This refers to the step size control algorithm, and the value of this parameter can be anything from 0.0 to 1.0. 1.0 is the most aggressive. In practice, try starting with this set to 1.0. If the solver fails, then reset to a smaller number.

### Natural Parameter Homotopy

Figure 8.2 contains a natural parameter homotopy netlist. It is the same circuit as was used in figure 8.1, except that some of the parameters are different. As before, the lines pertinent to the homotopy algorithm are highlighted in red.

#### Explanation of Parameters, Best Practice

There are a few differences between the netlist in figure 8.1 and figure 8.2. They are:

- **continuation=1.** Sets the algorithm to use natural parameter homotopy.

- **conparam=VDDdev.** If using natural parameter homotopy, this is required. It sets which input parameter to perform continuation on. The parameter name is subject to the same rules as parameter used by the .STEP capability. (See section 7.4). In this case the parameter is the magnitude of the DC voltage source, VDDdev. For this type of voltage source, it was possible to use the default device parameter (see section 7.4).
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3

* HOMOTOPY Options
.options device voltlim=0
.options nonlin continuation=1
.options loca stepper=0 predictor=0 stepcontrol=1
+ conparam=VDDdev
+ initialvalue=0.0 minvalue=-1.0 maxvalue=5.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200

VDDdev VDD 0 5V
RIN IN 1 1K
VIN1 1 0 5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1 VOUT 0 10K
C2 VOUT 0 0.1p
MN1 VOUT IN 0 0 CD4012_NMOS L=5u W=175u
MP1 VOUT IN VDD VDD CD4012_PMOS L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END

Figure 8.2. Example natural parameter homotopy netlist.
Using the magnitudes of independent voltage and current sources is a fairly obvious approach. Unfortunately, it doesn’t seem to work very well in practice.
9. Results Output and Evaluation Options

Chapter Overview

This chapter illustrates how to output simulation results to data or output files.

- Section 9.1 Control of Results Output
- Section 9.2 Additional Output Options
- Section 9.3 Evaluating Solution Results
9.1 Control of Results Output

_Xyce_ supports only one solution output command, `.PRINT`. `.PRINT` is quite flexible, and supports several output formats.

**.PRINT Command**

The `.PRINT` command sends the analysis results to an output file. _Xyce_ supports several options on the `.PRINT` line of netlists that control the format of the output. The syntax for the command is as follows:

```
».PRINT <analysis type> [options] <output variable(s)>
```

**Example:**

```
».PRINT TRAN FILE=output.prn V(3) V(2) V(4)
».PRINT DC format=tecplot FILE=Output.dat V(2) \{abs(V(4))*5.0\}
```

Table 9.1 gives the various options currently available to the `.PRINT` command. Note that as of _Xyce_ Release 2.1, it is possible to put expressions on the `.PRINT` line. For further information, see the _Xyce_ Reference Guide [3].

9.2 Additional Output Options

**.OPTIONS OUTPUT Command**

The main purpose of the `.OPTIONS OUTPUT` command is to provide control of the frequency at which data is written to files specified by `.PRINT TRAN` commands. This can be especially useful in controlling the size of the results file for simulations which required a large number of time steps. An additional benefit is that reducing the output frequency from the default, which outputs results at every time-step, can improve performance. The format for controlling the output frequency is:

```
».OPTIONS OUTPUT INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]
```

where `INITIAL_INTERVAL=<interval>` specifies the starting interval time for output and `<tx ix>` specifies later simulation times (tx) where the output interval will change to (ix).

The following example shows the output being requested (via the netlist `.OPTIONS OUTPUT` command) every 0.1\(\mu\)s for the first 10\(\mu\)s, every 1\(\mu\)s for the next 10\(\mu\)s, and every 5\(\mu\)s for the remainder of the simulation:
### 9.2 Additional Output Options

<table>
<thead>
<tr>
<th>Option...</th>
<th>Action...</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT=&lt;STD</td>
<td>NOINDEX</td>
</tr>
<tr>
<td>FILE=&lt;output filename&gt;</td>
<td>Allows the user to specify the output filename. The default is the netlist filename with the characters “.prn” appended (e.g., foo.cir.prn where foo.cir was the input netlist filename.</td>
</tr>
<tr>
<td>WIDTH=&lt;print field width&gt;</td>
<td>Allows the user to control the column width for the output data.</td>
</tr>
<tr>
<td>PRECISION=&lt;floating point precision&gt;</td>
<td>Controls the number of significant digits past the decimal point.</td>
</tr>
<tr>
<td>FILTER=&lt;filter floor value&gt;</td>
<td>Specifies the absolute value below which output variables will be printed as 0.0.</td>
</tr>
<tr>
<td>DELIMITER=&lt;TAB</td>
<td>COMMA&gt;</td>
</tr>
</tbody>
</table>

**Table 9.1.** .PRINT command options.
Example:  

```
.OPTIONS OUTPUT INITIAL_INTERVAL=.1us 10us 1us 20us 5us
```

Note: Xyce will output data at the next time that is greater-than or equal to the current interval time. This means that output might not correspond exactly to the time intervals due to the adaptive time stepping algorithm.

### 9.3 Evaluating Solution Results

This section describes how to view graphical waveform analysis of the simulation results generated by Xyce. You can use the solution output features of Xyce in conjunction with graphing tools (e.g., TecPlot, gnuplot, MS Excel, etc.) to analyze graphically the waveform data created by a Xyce circuit simulation (see Figure 9.1 below for an example plot using TecPlot, http://www.amtec.com). In addition, by using the FORMAT=PROBE option to the .PRINT command, Xyce is able to output .csd files which can be read by the PSpice Probe utility to view the results. See the PSpice Users Guide [4] for instructions on using the Probe tool, and the Xyce Reference Guide [3] for details on the options to the .PRINT command.

![Figure 9.1. TecPlot plot of diode clipper circuit transient response from Xyce .prn file.](image)

Xyce produces two types of output: the simulation output file and the waveform data file.
The calculations and results reported in the simulation output file can be thought of as an audit trail of the simulation. However, graphical analysis of data in the waveform data file is the most useful and accommodating way to evaluate simulation results.
10. Guidance for Running Xyce in Parallel

Chapter Overview

This chapter gives guidance on how to run a parallel version of Xyce. It includes the following sections:

- Section 10.1 Introduction
- Section 10.2 Mechanics
- Section 10.3 Problem Size
- Section 10.4 Linear Solver Options
- Section 10.5 Partitioning Options
10.1 Introduction

*Xyce* has been designed, from the ground up, to support a message-passing parallel implementation. As such, *Xyce* is unique among circuit simulation tools, and many of the issues pertinent to running in parallel are still research issues. However, *Xyce* is now mature enough that some general principles have emerged, for effectively running problems in a parallel environment.

10.2 Mechanics

Parallel simulations must be run from the command line. Details of how to do this are given in section 2.2.

10.3 Problem Size

Due to the overhead of interprocessor communication, running *Xyce* in parallel is only useful for large circuit problems. Also, for any problem size, there is an optimal number of processors. As one increases the processor count, the amount of communication required increases and the work per processor decreases. This increase in communication will slow a simulation down, while the reduction in work per processor will have the reverse effect. If the number of processors is too large, the benefit of distributing the problem will be outweighed by the high cost of communication overhead. Such a limit exists for every size of problem, and increasing the processor count beyond this point is counterproductive. This issue is most pronounced for platforms with a high communication cost, such as Beowulf clusters.

Smallest Possible Problem Size

Circuits are comprised of a discrete set of components (voltage nodes, devices, etc.). To run in parallel, it is preferable that *Xyce* be able to put at least one discrete part of the problem on each processor. In practice, this means that the number of processors should be less than the number of nodes in the circuit. *Xyce* is capable of simulating smaller problems, but it is not recommended, due to solver instabilities.

Ideal Problem Size

To take full advantage of *Xyce*’s parallel capability, the problem should be relatively large. A good metric for estimating how many processors one should use is the number of devices per processor. The ideal number of devices per processor is machine and problem
dependent. For machines such as SGI Challenger platforms, with relatively fast communication speeds in comparison to processor performance, reasonable speedups can be seen for 100’s of devices per processor. For Beowulf clusters with relatively slow communication speeds in comparison to processor performance, 1000’s of devices per processor are required to achieve reasonable speedups in parallel. These numbers are problem-dependent, as the effectiveness of the load balance and partitioning can vary substantially for different circuits.

10.4 Linear Solver Options

There are several different linear solvers available in Xyce version 2.1. They are:

- The AztecOO iterative solver library.
- SuperLU.
- Kundert Sparse (KSparse).
- KLU

For Xyce version 2.1, AztecOO is the only fully parallel solver available. However, both SuperLU and KLU are available with the parallel version of Xyce. With the later two solvers, the problem will be "assembled" in parallel, but the linear system will be solved in serial on processor 0. This can be quite effective for smaller parallel problems of a few thousand devices or less. KSparse is only available in serial Xyce.

10.5 Partitioning Options

Xyce currently has graph partitioning available. This partitioning subdivides the circuit problem into sections that are then distributed to the processors. A good partition can have a dramatic effect on the parallel performance of circuit simulation. There are two key components to a good partition:

- Effective load balance.
- Minimizing communication overhead.

An effective load balance ensures that the computational load of the calculation is equally distributed among the available processors. Minimizing communication overhead seeks to distribute the problem in a way that reduces the impact of underlying message passing during the simulation run. For runs with a small number of devices per processor the
communication overhead becomes the critical issue, while for runs with larger numbers of
devices per processor the load balancing becomes more important. Xyce has integrated
within it the ZOLTAN library of parallel partitioning heuristics.

Zoltan Partitioning of the Linear System

Zoltan is accessible through the `.OPTIONS LINSOL` control line in the netlist. By adding
an options `TR_LOADBALANCE=1` to the linear options, the linear system is statically load
balanced based on the graph of the Jacobian matrix. The local system is also reordered
based on nested dissection which should improve conditioning and minimize fill. These
techniques can be very effective for improving the efficiency of the iterative linear solvers.
See the Zoltan User Guide [10] for details. This option is now used by default for parallel
problems.

Singleton Filtering of the Linear System

Singleton filtering refers to the reduction of the linear system through removal of all rows
and columns with single non-zero entries. The values associated with these removed
entries can be resolved as pre/post solve linear operations. A by-product of this reduction
is a more tractable, sparser linear system for both the load balancing and linear solver
algorithms. This functionality is turned on by adding the `TR_SINGLETON_FILTER=1` option
to the `.OPTIONS LINSOL` control line in the netlist.

An important caveat is that the performance and convergence of the linear solver for par-
allel problems can be substantially less robust for some circuits. This is a known issue
with parallel iterative solution of linear problems. If Xyce is not performing as expected for
these problems, please consult the developer team.
11. TCAD (PDE Device) Simulation with Xyce

Chapter Overview

This chapter gives guidance on how to use the mesh-based device simulation capability of Xyce. It includes the following sections:

- Section 11.1 Introduction
- Section 11.2 One Dimensional Example
- Section 11.3 Two Dimensional Example
- Section 11.4 Doping Profile
- Section 11.5 Electrodes
- Section 11.6 Meshing
- Section 11.7 Mobility Models
- Section 11.8 Bulk Materials
- Section 11.9 Solver Options
- Section 11.10 Output and Visualization
11.1 Introduction

This chapter describes how to use the mesh-based device simulation functionality of Xyce. This capability is based on the solution a coupled set of partial differential equations (PDEs), discretized on a mesh such as the one in Figure 11.1. Such devices are often referred to as TCAD devices, where TCAD stands for Technology Computer Aided Design. While the rest of Xyce is intended to be similar to analog circuit simulators such as SPICE, the TCAD device capability is intended to be similar to well-known device simulators such as PISCES [11] and DaVinci [12].

Two different TCAD devices are available Xyce: a one-dimensional device and a two-dimensional device. These two devices have been implemented in a manner which allows them to be invoked in the same way as a conventional lumped parameter circuit device. Generally, this capability is intended for very detailed simulation of semiconductor devices, such as diodes, bipolar transistors, and MOSFETs. One possible application of this capability is the evaluation and/or analysis of conventional SPICE-style lumped parameter models.

NOTE: As of Xyce Release 2.1, the TCAD devices should still be considered to be a beta-level capability. The primary focus of Xyce has been traditional analog circuit simulation, so the TCAD devices have not been subject to the same level of testing as the traditional, SPICE-style devices. The TCAD (PDE) device simulator in Xyce should be regarded as a prototype for Charon, a high performance 3D device simulator that is under development at Sandia.

NOTE: The use of square brackets [] in the doping and electrode specifications is no longer correct as of Xyce Release 2.1. The correct delimiter to use for these parameters is now the curly bracket {}.

Equations

The equations of device simulation are described by many references including Kramer [13] and Selberherr [14]. The most common formulation, and the one that is used in Xyce, is the drift-diffusion (DD) formulation. This formulation consists of three coupled PDE’s: a single Poisson equation for electrostatic potential and two continuity equations; one each for electrons and holes.

Poisson equation

The electrostatic potential $\phi$ satisfies Poisson’s equation:

$$- \nabla \cdot (\epsilon \nabla \phi(x)) = \rho(x)$$

(11.1)

where $\rho$ is the charge density and $\epsilon$ is the permittivity of the material. For semiconductor devices, the charge density is determined by the local carrier densities and the local
11.1 Introduction

Figure 11.1. MOSFET Mesh Example.

doping,

\[ \rho(x) = q(p(x) - n(x) + C(x)) \]  

(11.2)

Here \( p(x) \) is the spatially-dependent concentration of holes, \( n(x) \) the concentration of electrons, and \( q \) the magnitude of the charge on an electron. \( C(x) \) is the total doping concentration, which can also be represented as \( C(x) = N_D^+(x) - N_A^-(x) \), where \( N_D^+ \) is the concentration of positively ionized donors, \( N_A^- \) the concentration of negatively ionized acceptors.

Species continuity equations

The continuity equations relate the convective derivative of the species concentrations to the creation and destruction of particles ("recombination/generation").

\[
\frac{\partial n(x)}{\partial t} + \nabla \cdot \Gamma_n = -R(x) \\
\frac{\partial p(x)}{\partial t} + \nabla \cdot \Gamma_p = -R(x) 
\]

(11.3) (11.4)

Here \( n \) is the electron concentration and \( p \) is the hole concentration. \( R \) is the recombination rate for both species. \( \Gamma_n \) and \( \Gamma_p \) are particle fluxes for electrons and holes, respectively. The sign of \( R \) is chosen because \( R \) is usually expressed as a recombination rate, and
is positive if particles are annihilating. The right hand sides are equal since creation and
destruction of carriers occurs in pairs.

One way in which the drift-diffusion model differs from other common formulations is the
manner in which the quantities $\Gamma_n$ and $\Gamma_p$ are determined. The expressions used are:

$$\Gamma_n = n(x) \mu_n E(x) + D_n \nabla n(x)$$  \hspace{1cm} (11.5)

$$\Gamma_p = p(x) \mu_p E(x) + D_p \nabla p(x)$$  \hspace{1cm} (11.6)

Here $\mu_n$, $\mu_p$ are mobilities for electrons and holes, and $D_n$, $D_p$ are diffusion constants.$E(x)$ is the electric field, which is given by the gradient of the potential, or $-\partial \phi / \partial x$.

**Discretization**

*Xyce* uses a box-integration discretization, with the Scharfetter-Gummel method for model-
ing the flux of charged species. This method has been described in detail elsewhere [13] [14] [15],
so it will not be described here.
11.2 One Dimensional Example

The one-dimensional device was the first PDE-based device to be implemented in Xyce. The single dimension limits its usefulness, but its simplicity makes it a good device to use for a preliminary example. One dimensional devices are almost always two-terminal diodes, and this fact allows for assumptions which simplify the specification and shorten the parameter list of the device.

An example netlist, for a simulation of a one-dimensional diode, is shown in Figure 11.2. The corresponding schematic is in Figure 11.3. The circuit is a regulator circuit, and is based on the principle that connecting one or more diodes in series with a resistor and a power supply will produce a relatively constant voltage. The input voltage (node 2) is a sinewave, with a frequency of 50 Hz and an amplitude of 1 V. The expected output (node 3) should be a (mostly) flat signal.

```
PDE Diode Regulator Circuit
VP 1 0 PULSE(0 5 0.0 2.0e-2 0.0 1.0e+20 1.2e+20)
VF 2 1 SIN(0 1 50 2.0e-2)
VT1 4 0 0V
R1 2 3 1k

* TCAD/PDE Device
Z1 3 4 DIODE na=1.0e19 nd=1.0e19 graded=0
 + l=5.0e-4 nx=101

 MODEL DIODE ZOD

.TRAN 1.0e-3 12.0e-2
.print TRAN format=tecplot
 + v(1) v(2) v(3) v(4) I(VF) I(VT1)

.options NONLIN maxstep=100 maxsearchstep=3
 + searchmethod=2 nox=0

.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
 + resettrannls=0

.END
```

Figure 11.2. One dimensional diode netlist. This circuit is a voltage regulator. The input signal should be a sinewave, while the output signal should be nearly flat. For the result of this netlist, see Figure 11.4.
Netlist Explanation

In Figure 11.3, the PDE device instance line is in red, while the PDE device model line is in blue. Currently, there are almost no model parameters for PDE devices. The model line serves only to set the level. The default level is 1, for a one-dimensional device. Two dimensional devices are invoked by setting level=2. Note that in this example, the level is not explicitly set, and so the default (1) is used.

The instance line is where most of the specific parameters are set for a TCAD device. In this example, the line appears as:

\[ \text{Z1 3 4 DIODE na=1.0e19 nd=1.0e19 graded=0 l=5.0e-4 nx=101} \]

\( \text{na} \) and \( \text{nd} \) are doping parameters, and represent the majority carrier doping levels on the N-side and the P-side of the junction, respectively. \( \text{graded}=0 \) is also a doping parameter, and specifies that the junction is not a graded junction, but is an abrupt step-function junction instead. \( l=5.0e-4 \) specifies the length of the device, in cm. \( \text{nx}=101 \) specifies that there are 101 mesh points, including the two endpoints. For the one-dimensional device, the mesh is always uniform, so the size of each mesh cell, \( \Delta x \) will be:

\[ \Delta x = \frac{l}{nx - 1} = \frac{5.0e-4 \text{ cm}}{100} = 5.0e-6 \text{ cm} \quad (11.7) \]

The mesh points \( i = 0 - 101 \) will have the following locations, \( x_i \):

\[ x_i = i \Delta x \]
\[ x_o = 0.0 \text{ cm} \]
\[ x_1 = 5.0e-6 \text{ cm} \]
11.2 One Dimensional Example

\[ x_{101} = 5.0 \times 10^{-4} \text{ cm} \]

Boundary Conditions and Doping Profile

Note that nothing has been specified in the example netlist about electrodes, or boundary conditions, and that the doping specification is minimal. This is because the example relies a lot on default parameters. A one dimensional device can only have exactly 2 electrodes connected to the circuit. These two electrodes are at opposite ends of the domain, one at the first mesh point \((x=0.0 \text{ cm}, i=0)\) and the other at the opposite end of the domain, at the last mesh point \((x=5.0 \times 10^{-4} \text{ cm}, i=101)\).

The electrode associated with the first mesh point \((x=0.0 \text{ cm})\) is connected to the second circuit node on the instance line, while the electrode associated with the last mesh point \((x=1)\) is connected to the first circuit node on the instance line. For the doping used in this example, the junction is in the exact center of the device \((x=1/2)\), and the n-side is the region defined by \(x<1/2\), and the p-side is the region defined by \(x>1/2\). This default doping, along with the electrode-circuit connectivity, result in the one-dimensional device to behave like a traditional SPICE-style diode. For a complete discussion of how to specify a doping profile see section 11.4. For a complete discussion of how to specify electrodes in detail (including boundary conditions), see section 11.5.

Results

The transient result of this circuit is shown in Figure 11.4. The input signal (node 2) is represented by the blue line, and the output signal (node 3) is represented by the red line. The voltage drop across the diode is nearly the same for a wide range of currents, and is approximately 0.67 V. The voltage drop across the series resistor, \(R_1\), is much more sensitive to the current magnitude, and so most of the voltage variation of the input sinewave is accounted for by \(R_1\).
Figure 11.4. Transient result for the voltage regulator circuit in Figure 11.2. The input voltage is represented by the blue line, while the output voltage is given by the red line.
11.3 Two Dimensional Example

An example netlist, for a simulation of a two-dimensional bipolar transistor, is shown in Figure 11.5. As before, the PDE device instance line is in red, while the PDE device model line is in blue. In this case, note that the level has been specified on the model line, and it has been set to 2. This is required for the two-dimensional device. This particular example is a DC sweep of a bipolar transistor device. A schematic, illustrating this circuit is shown in Figure 11.6.

Netlist Explanation

The two-dimensional device can have 2-4 electrodes. (this limitation will be relaxed in future versions of Xyce) In this example there are three; node 5, node 3 and node 7. These correspond to the three names on the "node" line, which appears as:

+ node = {name = collector, base, emitter}

This line specifies that node 5 is connected to an electrode named "collector", node 3 is connected to an electrode named "base", and node 7 is connected to an electrode named "emitter". Although this example only contains the electrode names, the "node" specification can contains a lot of information. For a full explanation of all the electrode parameters, see section 11.5.

The next line contains parameters concerned with plotting the results, and appears as follows:

+ tecplotlevel=2 txtdatalevel=1

Note that these are not related to the output specified by .PRINT, which outputs circuit data. The tecplotlevel command enables files to be output which are readable by tecplot. Tecplot can then be used to create contour plots of quantities such as the electron density, electrostatic potential and the doping profile. Figures 11.7 and 11.8 contain examples of tecplot-generated contour plots, which were generated from the results of this example.

The txtdatalevel command enables a text file with volume averaged information to be output to a file. Currently, both of these output files will be updated at each time step or DC sweep step.

The next line, mobmodel=arora, specifies which mobility model to use. For more detail on available mobility models, see section 11.7.

The last two lines, specify the mesh of the device, and are given by:

+ l=2.0e-3 w=1.0e-3
+ nx=30 ny=15
Two Dimensional Example

VPOS 1 0 DC 5V
VBB 6 0 DC -2V
RE 1 2 2K
RB 3 4 190K

Z1BJT 5 3 7 PDEBJT meshfile=internal.msh
+ node = {name = collector, base, emitter}
+ tecplotlevel=2 txtdatalevel=1
+ mobmodel=arora
+ l=2.0e-3 w=1.0e-3
+ nx=30 ny=15

* Zero volt sources acting as an ammeter to measure the
* base, collector, and emitter currents, respectively
VMON1 4 6 0
VMON2 5 0 0
VMON3 2 7 0

.MODEL PDEBJT ZOD level=2

.DC VPOS 0.0 12.0 0.5 VBB -2.0 -2.0 1.0
.options LINSOL type=superlu
.options NONLIN maxstep=70 maxsearchstep=1
  + searchmethod=2 in_forcing=0 nlstrategy=0
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
  + firstdcopstep=0 lastdcopstep=1

.PRINT DC V(1) I(VMON1) I(VMON2) I(VMON3)

.END

Figure 11.5. Two-dimensional BJT netlist. Some results of this netlist can be found in Figures 11.7 and 11.8.
This numbers are used in nearly the same way as the 1 and nx parameters were used in
the one-dimensional case. The mesh is Cartesian, and the spacing is uniform.

Doping Profile

As in the one-dimensional example, the two-dimensional example in figure 11.5 does not
specify anything about the doping profile, and thus relies upon defaults. In this case there
are three specified electrodes, which by default results in the doping profile of the bipolar
junction transistor (BJT). For a complete description of how to specify a doping profile in
detail, see section 11.4. This section also describes the various default impurity profiles.

Boundary Conditions and Electrode Configuration

As in the one-dimensional example, the two-dimensional example in figure 11.5 does not
specify anything about the electrode configuration or the boundary conditions, and relies
on default settings. To be consistent with the default 3-terminal doping, the device has
terminals that correspond to that of a BJT. All three electrodes (collector, base, emitter) are
along the top of the device.

By default all electrodes are considered to be neutral contacts. The boundary conditions
applied to the electron density, hole density and electrostatic potential are all Dirichlet
conditions.

For a complete discussion of how to specify electrodes in detail (including boundary con-
ditions), see section 11.5.

Results

Results for the two-dimensional example can be found in Figures 11.7, 11.8 and 11.9. The
first two figures are contour plots of the electrostatic potential. The first one corresponds
to the first DC sweep step, where VPOS is set to 0.0 Volts. The second one corresponds
to the final DC sweep step, in which VPOS has a value of 12.0 volts. The voltage source
VPOS applies a voltage to the emitter load resistor, RE, so some of the 12.0V is dropped
across RE, an the rest is applied to the BJT.

The third figure is an I-V curve of the dependence of the three terminal currents on applied
emitter voltage. For the entire sweep, a negative voltage of 2.0 V has been applied to the
base load resistor, and as this transistor is a PNP transistor, this results in the transistor
being in an “on” state. The emitter-collector current varies nearly linearly with the applied
emitter voltage. Also, the three currents sum to nearly zero, which one would expect
because of current conservation.

Note that the mesh is visible in Figure 11.7 and was generated using the internal “uniform
mesh” option. Generally using this sort of mesh will work numerically, in that Xyce will
converge to an answer. However, this mesh will probably not produce a very accurate result, as it does not resolve the depletion regions very well. In order to obtain better accuracy, either a finer uniform mesh would need to be used, or a nonuniform mesh, refined in around the depletion regions should be used. As described in section [11.6], refined, nonuniform meshes must be read in from an external mesh generator, such as the SGFramework [13].
Figure 11.6. Two-Dimensional BJT Circuit Schematic. This schematic is for the circuit described by the netlist in Figure 11.5. The mesh in the large circle is the mesh used in the example. The other mesh, which contains some mesh refinement, is included in the figure as an example of what is possible with an external mesh generator.
Figure 11.7. Initial Two-Dimensional BJT Result. Contour plot of the electrostatic potential at the first DC sweep step of the netlist in Figure 11.5. Note the mesh, which was generated using the internal “uniform mesh” option. This plot was generated using Tecplot.

Figure 11.8. Final Two-Dimensional BJT Result. Contour plot of the electrostatic potential at the last DC sweep step of the netlist in Figure 11.5. This plot was generated using Tecplot.
Figure 11.9. I-V Two-Dimensional BJT Result, for the netlist in Figure 11.5. The x-axis is in Volts. The three plotted currents are through the three BJT electrodes, and as expected they add (if corrected for sign) to zero. \(I(\text{VMON1})\) is the base current, \(I(\text{VMON2})\) the collector current, and \(I(\text{VMON3})\) the emitter current. \(V(1)\) is the voltage applied to the emitter load resistor, \(R_E\). This plot was generated using Tecplot.
11.4 Doping Profile

In the two examples, no doping parameters were specified, and Xyce used the defaults. Default profiles are uniquely specified by the number of electrodes. In practice, especially for two-dimensional simulations, the user will generally need to specify the doping profile manually.

**NOTE:** If an external mesh (from the SGFramework) is used, the doping profile will be read in from the mesh file, and it is not necessary (or appropriate) to specify any doping in the netlist.

**Manually Specifying the Doping**

A circuit netlist, which includes a one-dimensional device with a detailed, manual specification of the doping profile, is given in figure 11.10. A similar, two-dimensional, version of this problem is given in figure 11.12. For the purposes of this discussion, the one-dimensional example will be referred to, but information conveyed is equally applicable to the two-dimensional case.

In both examples, the parameters associated with doping are in red font. The doping is specified with one or more regions, which are summed together to get the total profile. Doping regions are specified in a tabular format, with each column representing a different region.

In the one-dimensional example, there are three regions, which are illustrated in figure 11.11. Region 1 is a uniform n-type doping, with a constant magnitude of 4.0e+12 donors per cubic cm. This magnitude is set by the parameter nmax. As the doping in this region is spatially uniform, the only meaningful parameters are function (which in this case specifies a spatially uniform distribution), type (ntype or ptype) and nmax. The others (nmin through flaty) are ignored for a spatially uniform region.

Region 2 is a more complicated region, in that the profile varies with space. This region is doped with p-type impurities, and has a Gaussian shape. Semiconductor processing often consists of an implant followed by an anneal, which results in a diffusive profile. The Gaussian function is a solution to the diffusion problem, when it is assumed that the impurity exists in a fixed quantity. Thus, the Gaussian shape is an appropriate choice for the doping regions of a lot of devices.

The peak of the Region 2 doping profile is given by the parameter nmin, and is 1.0e+19 acceptors per cubic cm. This peak has a location in the device which is specified by xloc=24.5e-4 cm. The parameters nmin and xwidth are fitting parameters.

Region 3 is also based on a Gaussian function, but unlike Region 2, it is flat on one side of the peak. This is set by the flatx parameter. The “flat” parameters follow the convention given by Table 11.1.
11.4 Doping Profile

Doping and Electrode specification example

TITLE Xyce PN Junction Simulation
vscope 0 1 0.0
rscope 2 1 50.0
cid 3 0 1.0u
r1 4 3 1515.0
vid 4 0 5.00
Z1DIODE 2 3 PDEDIODE nx=301 l=26.0e-4
* DOPING REGIONS: region 1, region 2, region 3
+ region= {function = uniform, gaussian, gaussian
+ type = ntype, ptype, ntype
+ nmax = 4.0e+12, 1.0e+19, 1.0e+18
+ nmin = 0.0e+00, 4.0e+12, 4.0e+12
+ xloc = 0.0 , 24.5e-04, 9.0e-04
+ xwidth = 0.0 , 4.5e-04, 8.0e-04
+ flatx = 0 , 0 , -1 }
*--------end of Diode PDE device ----------------

.MODEL PDEDIODE ZOD level=1
.options LINSOL type=superlu
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END

Figure 11.10. One-dimensional example, with detailed doping.
Figure 11.11. Doping Profile, Absolute Value. This corresponds to the doping specified by the netlist in figure 11.10.
11.4 Doping Profile

**Table 11.1: Description of the flatx, flaty doping parameters**

<table>
<thead>
<tr>
<th>flatx or flaty value</th>
<th>Description</th>
<th>1D Cross Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Gaussian on both sides of the peak (xloc) location.</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Gaussian if x&gt;xloc, flat (constant at the peak value) if x&lt;xloc.</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>Gaussian if x&lt;xloc, flat (constant at the peak value) if x&gt;xloc.</td>
<td></td>
</tr>
</tbody>
</table>

**Default Doping Profiles**

*Xyce* has a few default doping profiles which are invoked if the user doesn’t bother to specify detailed doping information. The default doping profiles are an artifact of early TCAD device development in *Xyce*, but are sometimes still useful. In particular, the simple step-junction diode is often a useful canonical problem. It is convenient to invoke a step junction doping without having to use the more complex region tabular specification.

Most real devices will have doping profiles that do not exactly match the default profiles. When attempting to simulate a realistic device, it will be necessary to skip the defaults and use the region tables described in the previous section.

**One Dimensional Case**

For the one-dimensional case, it is assumed that the doping profile is that of a simple junction diode, with the junction location exactly in the middle. The acceptor and donor concentrations are given by the parameters Na and Nd, respectively.

Note that the usage of Na and Nd, implicitly specifies a step junction doping profile, and is mutually exclusive with the more complex “doping region” table specification, described in section 11.4. If a netlist is input to *Xyce* which includes both a region table and Na (or Nd), the code will immediately exit with an error.

**Two Dimensional Case**

Doping level defaults in the two dimensional case are somewhat more complicated than in the one-dimensional case, because having two-dimensions allows for more configurations, and an arbitrary number (2-4) of electrodes. In *Xyce*, it was decided that the default doping profiles would be determined uniquely by the number of electrodes. The three available
default dopings are given in Table 11.2. In the case of the BJT and MOSFET dopings, it is possible to specify either n-type or p-type using the \texttt{type} instance parameter. If the detailed, manual doping is used, then the \texttt{type} parameter is ignored.

For a two-electrode device, the default doping is that of a simple diode. The acceptor and donor doping parameters, \( N_a \) and \( N_d \) are used in the same manner as in the one-dimensional device. As in the one-dimensional device, the junction is assumed to be exactly in the middle of the domain.

For a three-electrode device (like the example), the default doping is that of a bipolar junction transistor (BJT). By default the transistor is a PNP, but by setting the instance parameter \texttt{type=NPN}, an NPN transistor can be specified instead. The two-dimensional example in section 11.3 relies on this default.

For a four-terminal device, the default doping is that of a metal-oxide-semiconductor (MOSFET). Currently, the maximum number of electrodes is four, and no default profiles are available for more than four electrodes. By default this transistor is assumed to be NMOS, rather than PMOS.

<table>
<thead>
<tr>
<th>Number of Electrodes</th>
<th>Doping Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Step Function Diode</td>
</tr>
<tr>
<td>3</td>
<td>Bipolar Junction Transistor (BJT)</td>
</tr>
<tr>
<td>4</td>
<td>Metal-Oxide Semiconductor</td>
</tr>
<tr>
<td></td>
<td>Field-Effect Transistor(MOSFET)</td>
</tr>
</tbody>
</table>

Table 11.2: Default Doping profiles for different numbers of electrodes
11.5 Electrodes

In the two examples, minimal electrode were specified, and Xyce used the defaults. In practice, especially for two-dimensional simulations, the user will need to specify the electrodes in more detail.

**NOTE:** If an external mesh (from the SGFramework) is used, some of the electrode information (the locations, and lengths) will be specified in the mesh file, so they should not be specified in the netlist.

Manually Specifying the Electrodes

A detailed electrode specification is specified in blue font in Figure 11.12. As with the doping parameters, the electrode parameters are specified in a tabular format, in which each columns of the table specifies the parameters for a different electrode. The most important parameter (for getting the code to run without immediately exiting with an error) is the name parameter. It is the only required parameter.

The number of specified electrodes must match the number of connected circuit nodes, and the order of the electrode columns, from left to right, is in the same order as the circuit nodes, also from left to right. In the example of Figure 11.12, the first electrode column, which specifies an electrode named “anode”, is connected to the circuit through circuit node 2. Respectively, the second column, for the “cathode” electrode, is connected to the circuit by circuit node 3.

If using an external mesh (see section 11.6), the external mesh file must have this same number of electrodes as well. Also, if using the external mesh, the electrode names specified in the electrode table must match (case insensitive) with the electrode names used by the external mesh.

Boundary Conditions

In the example, the bc parameter has been set to “Dirichlet” on all the electrodes, which is the default. The bc parameter sets the type of boundary condition that is applied to the density variables, the electron density and the hole density. There are two possible settings for the bc parameter, Dirichlet and Neumann. If Dirichlet is specified, the electron and hole densities are set to a specific value at the contact, and the applied values enforce charge neutrality. See the Xyce Reference Guide for the charge-neutral equation [3]. If Neumann is specified, a zero-flux condition is applied, which enforces that the current through the electrode will be zero.

This parameter does not affect the electrostatic potential boundary condition. The boundary condition applied to the potential is always Dirichlet, and is (in part) determined from
Doping and Electrode specification example

vscope 1 0 0.0
rscope 2 1 50.0
cid 3 0 1.0u
r1 4 3 1515.0
vid 4 0 1.00

*------------- Diode PDE device ------------------
Z1DIODE 2 3 PDEDIODE
+ tecplotlevel=1 txtdatalevel=1 cyl=1
+ meshfile=internal.msh
+ nx=25  l=70.0e-4 ny=40  w=26.0e-4

* ELECTRODES:  ckt node 2, ckt node 3
+ node = {name = anode, cathode
+ bc = dirichlet, dirichlet
+ start = 0.0, 0.0
+ end = 70.0e-4, 70.0e-4
+ side = top, bottom
+ material = neutral, neutral
+ oxideBndryFlag = 0, 0

* DOPING REGIONS:  region 1, region 2, region 3
+ region= {function = uniform, gaussian, gaussian
+ type = ntype, ptype, ntype
+ nmax = 4.0e+12, 1.0e+19, 1.0e+18
+ nmin = 0.0e+00, 4.0e+12, 4.0e+12
+ xloc = 0.0, 60.0e-04, 100.0
+ xwidth = 0.0, 4.0e-04, 1.0
+ yloc = 0.0, 24.5e-04, 9.0e-04
+ ywidth = 0.0, 4.5e-04, 8.0e-04
+ flatx = 0, -1, -1
+ flaty = 0, 0, -1

*--------end of Diode PDE device ----------------

.MODEL PDEDIODE ZOD  level=2
.options LINSOL type=superlu
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END

Figure 11.12. Two-dimensional example, with detailed doping and detailed electrodes.
the connected nodal voltage. To apply a specific voltage to an electrode contact, a voltage source should be attached to it, such as \( V_{BB} \) in the schematic Figure 11.6.

**Electrode Material**

Several different electrode materials can be specified. A list is given in Table 11.3. The main effect of any metal (non-neutral) material is the impose a Schottky barrier at the contact. This generally makes numerical solution more difficult, so any materials should be applied with caution.

The *Xyce* Reference Guide [3] has a detailed description of Schottky barriers and how they are imposed on contacts in *Xyce*. Also, values for electron affinities of various bulk materials and workfunction values for the various metal contacts are given in the Reference Guide.

<table>
<thead>
<tr>
<th>Material</th>
<th>Symbol</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>neutral</td>
<td>neutral</td>
<td>Default</td>
</tr>
<tr>
<td>aluminum</td>
<td>al</td>
<td></td>
</tr>
<tr>
<td>p+-polysilicon</td>
<td>ppoly</td>
<td></td>
</tr>
<tr>
<td>n+-polysilicon</td>
<td>npoly</td>
<td></td>
</tr>
<tr>
<td>molybdenum</td>
<td>mo</td>
<td></td>
</tr>
<tr>
<td>tungsten</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>molybdenum disilicide</td>
<td>modi</td>
<td></td>
</tr>
<tr>
<td>tungsten disilicide</td>
<td>wdi</td>
<td></td>
</tr>
<tr>
<td>copper</td>
<td>cu</td>
<td></td>
</tr>
<tr>
<td>platinum</td>
<td>pt</td>
<td></td>
</tr>
<tr>
<td>gold</td>
<td>au</td>
<td></td>
</tr>
</tbody>
</table>

Table 11.3: Electrode Material Options. Neutral contacts are the default, and pose the least problem to the solvers.

There is also an oxideBndryFlag parameter, which if set to true (1), will model the contact as having an oxide layer in between the metal contact and the bulk semiconductor. Note that this oxide layer model does not currently include displacement current, so transient capacitive effects will not be seen in the results.

**Location Parameters**

Each electrode has three location parameters: start, end, and side. These are only necessary if using the internal mesh and should not be specified if using an external, SGFramework mesh.
For the internal mesh, the mesh is assumed to be rectangular, and any electrode is assumed to be on one of the four sides. The four side possibilities are: top, bottom, right and left. These four sides are parallel to mesh directions. The start and end parameters are floating point numbers which specify the starting and ending location of an electrode, in units of cm.

The lower left hand corner of the mesh rectangle is located at the origin. A side=bottom electrode with start=0.0 and end=1.0e-4 will originate at the lower left hand corner of the mesh (x=0.0, y=0.0) and end at (x=1.0e-4, y=0.0).

**NOTE:** *Xyce* will attempt to match the specified electrode to the specified mesh. However, if the user specifies a mesh that is not consistent with the electrode locations, the electrodes will not be able to have the exact length specified. For example, if the mesh spacing is $\Delta x = 1.0e-5$, then the electrodes can only have a length that is a multiple of 1.0e-5.

### Electrode Defaults

There are defaults for all the electrode parameters except the names. In practice, the locations of the electrodes will usually be explicitly specified (either using the electrode table, or as part of an external mesh file). Default electrode locations have been created to correspond with the default dopings, and they should only be used in that context.

#### Location Parameters

In practice, the locations of the electrodes will usually be explicitly specified, but they have defaults to correspond with the default dopings. The default electrode locations in one-dimensional devices are for that of a diode. One electrode is located at $x=x_{\text{min}}$, while the other is located at $x=x_{\text{max}}$.

The default electrode locations in two-dimensional devices are dependent on the number of electrodes, similar to the default dopings. Table 11.2 can be used to determine the configurations. For the two-terminal diode, the two electrodes are along the y-axis, at the $x=x_{\text{min}}$ and $x=x_{\text{max}}$ extrema. For the three-terminal BJT, all three electrodes are parallel to the x-axis, along the top, at $y=y_{\text{max}}$. For the four-terminal MOSFET, the drain, gate, and source electrodes are also along the top, but the bulk electrode spans the entire length of the bottom of the mesh, at $y=y_{\text{min}}$.

#### Other Parameters

The default contact material is neutral. The default oxideBndryFlag is false (0). The default boundary condition (bc) is Dirichlet.
11.6 Meshes

Meshes from the SG Framework (External, 2D)

It is possible to have Xyce read in a two-dimensional mesh which was generated externally, by the SGFramework [13]. The mesh pictured in Figure 11.1 is such a mesh, and so is the refined mesh (not inside the circle) in Figure 11.6. To use an SGF-generated mesh, the instance parameter, "meshfile" must be used, and set to be the name of the SGFramework-generated file. Xyce will assume that the mesh file is located in the local execution directory. One advantage of using an externally generated mesh (over an internally generated mesh - see next section) is that external meshing tools are more sophisticated, and in particular have mesh refinement capabilities.

Instructions for the usage of the SGFramework is outside the scope of this document. If the user wishes to generate meshes in this manner, it is best to consult Kramer [13]. Future versions of Xyce may accept mesh files generated by other mesh generators, such as Cubit [16].

Cartesian Meshes (Internal, 1D and 2D)

One dimensional and two-dimensional devices can both create Cartesian meshes, without requiring an external mesh generator. For the two-dimensional devices, it is necessary to specify meshfile=internal.msh to invoke the Cartesian meshing capability. For one-dimensional devices, this isn’t needed, as there is no other option.

Meshes generated in this manner are very simple, in that there are only two parameters per dimension, and the resulting mesh is uniform. An example of such a mesh can be seen in Figure 11.7. The mesh spacing is determined from the following expressions:

\[ \Delta x = \frac{l}{nx - 1} \tag{11.8} \]
\[ \Delta y = \frac{w}{ny - 1} \tag{11.9} \]

This mesh specification assumes that the domain is a rectangle. Non-rectangular domains can only be described using an external mesh program.

Cylindrical meshes, 2D

For two-dimensional devices, the simulation area may be a cylinder slice. This capability is turned on by the instance parameter, cyl=1. For an example, see Figure 11.13. It is
assumed that the axis of the cylinder corresponds to the minimum radius (or x-axis value) of the mesh, while the circumference corresponds to the maximum radius (or maximum x-axis value). This feature can be applied to either external or internal two-dimensional meshes.

![Cylindrical Mesh Example](image)

**Figure 11.13.** Cylindrical Mesh Example. This mesh has been designed to match the electron microscope image, which is of a stockpile device.
11.7 Mobility Models

There are several mobility models available to both the one and two dimensional devices, and they are listed in Table 11.4. These models are fairly common, and can be found in most device simulators. These models are described in more detail in the Xyce Reference Guide.

<table>
<thead>
<tr>
<th>Mobility Name</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>arora</td>
<td>Basic mobility model</td>
<td>Arora, et al. [17]</td>
</tr>
<tr>
<td>analytic</td>
<td>Basic mobility model</td>
<td>Caughy and Thomas [18]</td>
</tr>
<tr>
<td>carr</td>
<td>Includes carrier-carrier interactions</td>
<td>Dorkel and Leturq [19]</td>
</tr>
</tbody>
</table>

Table 11.4: Mobility models available for PDE devices

Specifying the mobility model from the netlist is done by setting the `mobmodel` parameter to the name of the model. Model names are given in the first column of Table 11.4. The mobility model is specified as an instance parameter on the device instance line, as (typically) `mobmodel=arora`. See the usage in Figure 11.5 for a more detailed example.

The default mobility is the "carr" mobility, which includes carrier-carrier interactions. This model has a stronger dependence on carrier density than the other two models, and introduces some nonlinearity into the problem. If having convergence problems, consider using either the "arora" or "analytic" model, as both of these models are a little bit simpler.
11.8 Bulk Materials

The bulk material is specified using the `bulkmaterial` instance parameter. Xyce currently supports Silicon (`si`) as a bulk material and this is the default. It can also simulate Gallium Arsenide (`gaas`) and Germanium (`ge`), but these materials have not been extensively tested.

The mobility models described in the previous section each support all three materials, and the dielectric permittivity is correct for all three, but the carrier lifetime models may not be. These issues will be resolved in a future Xyce release.
11.9 Solver Options

Problems that are based on TCAD/PDE devices have different optimal solver settings than do analog circuit problems. Generally, as these devices are mesh-based, and have a more predictable topology, iterative linear solvers have a better chance of being successful than they do for analog circuit simulation. Note that if using a direct solver (klu, superlu, or ksparse), the best option is superlu, because ksparse and klu have both been optimized for circuits (not mesh-based PDE problems), while superlu is more general.

On the nonlinear solver level, voltage limiting doesn’t have an obvious application to PDE devices, and quadratic line search appears to be the best algorithm. The solver options specified in the example netlist Figure 11.5 are adequate for simulations that have a simple (linear) circuit attached.

For problems which involve a complicated external circuit, it is best to apply the two-level Newton algorithm to the nonlinear solve. This algorithm is described in detail in Keiter [20] and Mayaram [21]. While this algorithm has been implemented and exercised within Xyce, it is not part of the Release 2.1 version. To use this algorithm, the user will need to obtain a special “custom” build of Xyce, or wait until a future release.
11.10 Output and Visualization

Using the .PRINT Command

For simple plots (such as I-V curves), output results for Xyce can be generated with the .PRINT statement, which is described in detail in section 9.1. Figures 11.4 and 11.9 are examples of the kind of data that is produced with .PRINT statement netlist commands. These particular figures were plotted in Tecplot, but many other plotting programs would also have worked, including XDAMP [22].

Multi-dimensional Plots

Device simulation has visualization needs which go beyond that of conventional circuit simulation. Multi-dimensional perspective and/or contour plots are often desirable. Xyce is capable of outputting multi-dimensional plot data in several formats, including Tecplot, GnuPlot, and sgplot. Currently, the options for each of these formats can only enable or disable the output of files, and when enabled, a new file (or a new append to an existing file) will happen at every time step or DC sweep step. For long simulations, this may produce a prohibitive number of files. Currently, there is no equivalent to the .OPTIONS OUTPUT INITIAL_INTERVAL command, nor does the output of plot data currently use this command. Plot files are either output at every step or not at all.

For each type of plot file, the file is placed in the execution directory. Each individual device instance is given a unique file, or files, and the file names are derived from the name of the PDE device instance. The instance names provides the prefix, and the file type (tecplot, gnuplot, sgplot) determines the suffix.

Tecplot Data

Tecplot is a commercial plotting program from Amtec Engineering, Inc., and is the best choice for creating contour plots of spatially dependent data. All of the graphical examples in this chapter were created with Tecplot. (see Figures 11.7 and 11.8 for examples) The output of Tecplot files is enabled using the instance parameter, tecplotlevel=1. If set to zero, no tecplot files are output. If set to one, a separate tecplot file is output for each nonlinear solve. If set to two, a single tecplot file, which contains data for every nonlinear solve is created and is appended at the end of each solve.

By default tecplotlevel is set to one, meaning the code will, by default produce a separate Tecplot file for each nonlinear solve. The suffix for Tecplot data files is *.dat. Internally, the file is an ASCII text file. Tecplot does have a binary format, but Xyce has not yet been set up to use it.

Note that it is also possible to set tecplotlevel=2. Doing this will force Xyce to create one
single tecplot file, and the data from each solve will be appended to this file as a separate zone. This makes it possible to use Tecplot to create animations.

**Gnuplot Data**

Gnuplot is an open source plotting program, which is available on most Linux/Unix platforms. The parameter for this type of output is `gnuplotlevel=1`. This type of output file is off (zero) by default, meaning no gnuplot files will be output. The suffix for Gnuplot files is *Gnu.dat. Like tecplot files, Gnuplot files are also in ASCII text format.

**NOTE:** Gnuplot will only work with structured Cartesian meshes. Externally created, unstructured meshes (even ones that appear Cartesian) cannot be plotted with Gnuplot.

**Sgplot Data**

Sgplot is the plotting program for the SGFramework [13]. The parameter for this type of output is `sgplotlevel=1`. This type of output file is off (zero) by default. The suffix for Sgplot data files is *.res. Internally this file is in binary format. Note that it is not a machine-independent file format.

**Volume Averaged Data**

*Xyce* can also output volume-averaged information for each PDE device. This is enabled by setting the instance parameter, `txtdatalevel=1`. It is off (zero) by default, meaning no text files with volume averaged data will be output.
This page is left intentionally blank
Bibliography


Index

Xyce

running, 32
running in parallel, 34, 99
.DC, 45
.INCLUDE, 55
.MODEL, 55
.OP, 78
.OPTIONS
    LINSOL, 102
    OUTPUT, 76, 94
    RESTART, 76, 77
.PRINT, 34, 95
    DC, 45
    FORMAT, 96
    TRAN, 47, 94
.STEP, 81
.SUBCKT, 55
.TRAN, 47, 74
runxyce, 31, 32
runxyce.bat, 31
xmpirun, 31, 32

analog behavioral modeling (ABM), 58, 67
analysis
    DC, 77
    DC sweep, 45, 77
    STEP, 81
    transient, 47, 74

behavioral model, 55, 58
    analog behavioral modeling (ABM), 58
    67, 68
    lookup table, 69
    bias point, 74, 77
    bifurcation, 87
    checkpoint, 76

format, 76
ChileSPICE, 19
circuit
    elements, 52
    simulation, 52
    topology, 52, 53
command line, 32, 34
    options, 33
    output, 33
comments in a netlist, 54
continuation, 87

DC analysis, 77
DC Sweep, 77
DC sweep, 45
    OP Analysis, 78
    running, 78
device
    B (nonlinear dependent) source, 68
    analog, 54, 55
    analog device summary, 56
    B source, 56
    behavioral, 68
    behavioral model, 55
    bipolar junction transistor (BJT), 56
    capacitor, 55
    device types, 55
diode, 56
    independent current source, 56
    independent voltage source, 56
    inductor, 55
    instance, 55
    JFET, 56
    MOSFET, 56
    mutual inductor, 56
    nonlinear dependent source, 56

137
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDE Devices</td>
<td>56</td>
</tr>
<tr>
<td>resistor</td>
<td>55</td>
</tr>
<tr>
<td>specifying ABM devices</td>
<td>68</td>
</tr>
<tr>
<td>subcircuit</td>
<td>56</td>
</tr>
<tr>
<td>transmission line</td>
<td>56</td>
</tr>
<tr>
<td>voltage controlled current source</td>
<td>56</td>
</tr>
<tr>
<td>voltage controlled switch</td>
<td>56</td>
</tr>
<tr>
<td>voltage controlled voltage source</td>
<td>56</td>
</tr>
<tr>
<td>devices</td>
<td></td>
</tr>
<tr>
<td>PDE devices</td>
<td>103</td>
</tr>
<tr>
<td>TCAD devices</td>
<td>103</td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>checkpointing</td>
<td>76</td>
</tr>
<tr>
<td>circuit construction</td>
<td>42</td>
</tr>
<tr>
<td>DC sweep</td>
<td>45</td>
</tr>
<tr>
<td>declaring parameters</td>
<td>57</td>
</tr>
<tr>
<td>restarting</td>
<td>77</td>
</tr>
<tr>
<td>subcircuit model definition</td>
<td>63, 64</td>
</tr>
<tr>
<td>transient analysis</td>
<td>47</td>
</tr>
<tr>
<td>using expressions</td>
<td>59</td>
</tr>
<tr>
<td>using parameters</td>
<td>58</td>
</tr>
<tr>
<td>expressions</td>
<td>58</td>
</tr>
<tr>
<td>additional constructs for ABM modeling</td>
<td>69</td>
</tr>
<tr>
<td>arithmetic</td>
<td>60</td>
</tr>
<tr>
<td>example</td>
<td>59</td>
</tr>
<tr>
<td>lookup table</td>
<td>69</td>
</tr>
<tr>
<td>operators</td>
<td>59</td>
</tr>
<tr>
<td>time-dependent</td>
<td>69</td>
</tr>
<tr>
<td>using</td>
<td>58</td>
</tr>
<tr>
<td>valid constructs</td>
<td>58</td>
</tr>
<tr>
<td>graph partitioning</td>
<td>101</td>
</tr>
<tr>
<td>ground nodes</td>
<td>53</td>
</tr>
<tr>
<td>GUI</td>
<td>32</td>
</tr>
<tr>
<td>homotopy</td>
<td>87, 88</td>
</tr>
<tr>
<td>model</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>62</td>
</tr>
<tr>
<td>model organization</td>
<td>65</td>
</tr>
<tr>
<td>MPI</td>
<td>32, 33</td>
</tr>
<tr>
<td>netlist</td>
<td>42, 52</td>
</tr>
<tr>
<td>.END</td>
<td>52</td>
</tr>
<tr>
<td>.END statement</td>
<td>43</td>
</tr>
<tr>
<td>analog devices</td>
<td>54, 55</td>
</tr>
<tr>
<td>arithmetic expressions</td>
<td>60</td>
</tr>
<tr>
<td>command elements</td>
<td>54</td>
</tr>
<tr>
<td>comments</td>
<td>43, 54</td>
</tr>
<tr>
<td>device description</td>
<td>54</td>
</tr>
<tr>
<td>element</td>
<td>52</td>
</tr>
<tr>
<td>end line</td>
<td>54</td>
</tr>
<tr>
<td>expression operators</td>
<td>59</td>
</tr>
<tr>
<td>expressions</td>
<td>58</td>
</tr>
<tr>
<td>first line special</td>
<td>53</td>
</tr>
<tr>
<td>in-line comments</td>
<td>54</td>
</tr>
<tr>
<td>model definition</td>
<td>55</td>
</tr>
<tr>
<td>node names</td>
<td>53</td>
</tr>
<tr>
<td>parameters</td>
<td>57</td>
</tr>
<tr>
<td>restart</td>
<td>76</td>
</tr>
<tr>
<td>scaling factors</td>
<td>52</td>
</tr>
<tr>
<td>sources</td>
<td>74</td>
</tr>
<tr>
<td>subcircuit</td>
<td>55</td>
</tr>
<tr>
<td>title</td>
<td>43</td>
</tr>
<tr>
<td>title line</td>
<td>52, 53</td>
</tr>
<tr>
<td>using expressions</td>
<td>58</td>
</tr>
<tr>
<td>node names</td>
<td>53</td>
</tr>
<tr>
<td>OP analysis</td>
<td>78</td>
</tr>
<tr>
<td>output</td>
<td></td>
</tr>
<tr>
<td>.PRINT</td>
<td>94</td>
</tr>
<tr>
<td>.STEP</td>
<td>85</td>
</tr>
<tr>
<td>comma separated value</td>
<td>32</td>
</tr>
<tr>
<td>log file</td>
<td>32</td>
</tr>
<tr>
<td>specifying file name</td>
<td>32</td>
</tr>
<tr>
<td>time values</td>
<td>76</td>
</tr>
<tr>
<td>parallel</td>
<td></td>
</tr>
<tr>
<td>communication</td>
<td>101</td>
</tr>
<tr>
<td>computing</td>
<td>15, 16</td>
</tr>
<tr>
<td>distributed-memory</td>
<td>16</td>
</tr>
<tr>
<td>efficiency</td>
<td>16</td>
</tr>
<tr>
<td>graph partitioning</td>
<td>101</td>
</tr>
<tr>
<td>large scale</td>
<td>16</td>
</tr>
<tr>
<td>load balance</td>
<td>101</td>
</tr>
<tr>
<td>message passing</td>
<td>16</td>
</tr>
<tr>
<td>MPI</td>
<td>32, 33</td>
</tr>
<tr>
<td>number of processors</td>
<td>33</td>
</tr>
<tr>
<td>shared-memory</td>
<td>16</td>
</tr>
<tr>
<td>parallelGuidance</td>
<td>35</td>
</tr>
<tr>
<td>parameter</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Xyce™ Users’ Guide

declaring, 57
using in expressions, 58
PDE Device Modeling, 103
PDE Devices, 56
platforms
Apple/OSX, 33
HP/Compaq/Tru64, 33
Intel X86/FreeBSD, 33
Intel X86/Linux, 33
Intel X86/Microsoft Windows XP, 33
SGI/IRIX, 33
PSpice, 19, 42
Probe, 96
Reference Guide, 19
restart, 76, 77
format, 76, 77
results
evaluating, 96
output control, 94
output frequency, 94
output options, 93
print commands, 95
running Xyce, 32
runxyce, 32
Sandia National Laboratories, 15
schematic capture, 42
solvers
iterative linear, 102
transient, 75
sources, 74
defining time-dependent, 74
time-dependent, 75
waveforms, 75
SPICE, 42, 52
STEP parametric analysis, 81
subcircuit
hierarchy, 64
scope, 64
Synopsys
DaVinci, 18
Medici, 18
time step
maximum size, 76
size, 75
topology, 53
transient analysis, 47, 74
Unix, 19
Users of other circuit codes, 19
xmpirun, 32
ZOLTAN, 102
DISTRIBUTION:

1  Steven P. Castillo  
   Klipsch School of Electrical and 
   Computer Engineering  
   New Mexico State University  
   Box 3-o  
   Las Cruces, NM 88003  

1  Kwong T. Ng  
   Klipsch School of Electrical and 
   Computer Engineering  
   New Mexico State University  
   Box 3-o  
   Las Cruces, NM 88003  

1  Nick Hitchon  
   Electrical and Computer Engineering  
   University of Wisconsin  
   1415 Engineering Drive  
   Madison, WI 53706  

1  Mark Kushner  
   Department of Electrical and 
   Computer Engineering  
   University of Illinois  
   1406 W. Green Street  
   Urbana, IL 61801  

1  Ron Kielkowski  
   RCG Research, Inc  
   8605 Allisonville Rd, Suite 370  
   Indianapolis, In 46250  

1  Mike Davis  
   Software Federation, Inc.  
   211 Highview Drive  
   Boulder, Co 80304  

1  Wendland Beezhold  
   Idaho Accelerator Center  
   1500 Alvin Ricken Drive  
   Pocatello, Idaho 83201  

1  Kartikeya Mayaram  
   Department of Electrical and 
   Computer Engineering  
   Oregon State University  
   Corvallis, OR 97331-3211  

1  Linda Petzold  
   Department of Computer Science  
   University of California, Santa 
   Barbara  
   Santa Barbara, CA 93106-5070  

1  Jaijeet Roychowdhury  
   4-174 EE/CSci Building  
   200 Union Street S.E.  
   University of Minnesota  
   Minneapolis, MN 55455  

1  C.-J. Richard Shi  
   VLSI and Electronic Design Automation  
   210 EE/CSE Bldg.  
   Box 352500  
   University of Washington  
   Seattle, WA 98195  

1  Homer F. Walker  
   WPI Mathematical Sciences  
   100 Institute Road  
   Worcester, MA 01609  

1  Dan Yergeau  
   CISX 334  
   Via Ortega  
   Stanford, CA 94305-4075  

1  Masha Sosonkina  
   319 Heller Hall  
   10 University Dr.  
   Duluth, MN 55812  

1  Misha Elena Kilmer  
   113 Bromfield-Pearson Bldg.  
   Tufts University  
   Medford, MA 02155  

1  Tim Davis  
   P.O. Box 116120  
   University of Florida  
   Gainesville, FL 32611-6120
<table>
<thead>
<tr>
<th>Name</th>
<th>Company/Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achim Basermann</td>
<td>C&amp;C Research Laboratories, NEC Europe Ltd.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rathausallee 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D-53757 Sankt Augustin Germany</td>
<td></td>
</tr>
<tr>
<td>Philip A. Wilsey</td>
<td>Clifton Labs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7450 Montgomery Road</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Suite 300</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cincinnati, Ohio 45236</td>
<td></td>
</tr>
<tr>
<td>Dale E. Martin</td>
<td>Clifton Labs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7450 Montgomery Road</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Suite 300</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cincinnati, Ohio 45236</td>
<td></td>
</tr>
<tr>
<td>Lon Waters</td>
<td>CoMet Solutions, Inc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11811 Menaul Blvd NE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Suite No. 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Albuquerque, NM 87112</td>
<td></td>
</tr>
<tr>
<td>Tom Hunter</td>
<td></td>
<td>09000</td>
</tr>
<tr>
<td>Al Romig</td>
<td></td>
<td>01000</td>
</tr>
<tr>
<td>John Stichman</td>
<td></td>
<td>02000</td>
</tr>
<tr>
<td>Bill Camp</td>
<td></td>
<td>09200</td>
</tr>
<tr>
<td>Thomas C. Bickel</td>
<td></td>
<td>09100</td>
</tr>
<tr>
<td>Marion Scott</td>
<td></td>
<td>01700</td>
</tr>
<tr>
<td>Kenneth E. Washington</td>
<td></td>
<td>08900</td>
</tr>
<tr>
<td>Paul Yarrington</td>
<td></td>
<td>09230</td>
</tr>
<tr>
<td>Mike Knoll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robert Leland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sudip Dosanjh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paul V. Plunkett</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steven N. Kempka</td>
<td></td>
<td></td>
</tr>
<tr>
<td>John D. Zepper</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jaime L. Moya</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martin Pilch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anthony A. Giunta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stephen E. Lott</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark D. Rintoul</td>
<td></td>
<td></td>
</tr>
<tr>
<td>David Womble</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bruce Hendrickson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neil Pundit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Doug Doerfler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philip Heermann</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edward Boucheron</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS</td>
<td>0820</td>
<td>Patrick Chavez, 09232</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>John Aidun, 09235</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Scott A. Hutchinson, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Eric R. Keiter, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Deborah Fixel, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Robert J. Hoekstra, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Joseph P. Castro, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>David R. Gardner, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Gary Hennigan, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Roger Pawlowski, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Richard Schiek, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>1111</td>
<td>John N. Shadid, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>1111</td>
<td>Andrew Salinger, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Paul Lin, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Siriphone C. Kuthakun, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Eric L. Rankin, 09233</td>
</tr>
<tr>
<td>MS</td>
<td>0807</td>
<td>David N. Shirley, 9328</td>
</tr>
<tr>
<td>MS</td>
<td>0807</td>
<td>Philip M. Campbell, 9328</td>
</tr>
<tr>
<td>MS</td>
<td>0847</td>
<td>Scott Mitchell, 09211</td>
</tr>
<tr>
<td>MS</td>
<td>0847</td>
<td>Mike Eldred, 09211</td>
</tr>
<tr>
<td>MS</td>
<td>0847</td>
<td>Tim Trucano, 09211</td>
</tr>
<tr>
<td>MS</td>
<td>0847</td>
<td>Bart van Bloemen Waanders, 09211</td>
</tr>
<tr>
<td>MS</td>
<td>0196</td>
<td>Elebeoba May, 09212</td>
</tr>
<tr>
<td>MS</td>
<td>1110</td>
<td>Todd Coffey, 09214</td>
</tr>
<tr>
<td>MS</td>
<td>1110</td>
<td>David Day, 09214</td>
</tr>
<tr>
<td>MS</td>
<td>1110</td>
<td>Mike Heroux, 09214</td>
</tr>
<tr>
<td>MS</td>
<td>1110</td>
<td>James Willenbring, 09214</td>
</tr>
<tr>
<td>MS</td>
<td>1111</td>
<td>Karen Devine, 09215</td>
</tr>
<tr>
<td>MS</td>
<td>0310</td>
<td>Jim Ang, 09220</td>
</tr>
<tr>
<td>MS</td>
<td>1109</td>
<td>Robert Benner, 09224</td>
</tr>
<tr>
<td>MS</td>
<td>0822</td>
<td>Pat Crossno, 09227</td>
</tr>
<tr>
<td>MS</td>
<td>0822</td>
<td>David Rogers, 09227</td>
</tr>
<tr>
<td>MS</td>
<td>0316</td>
<td>Harry Hjalmarson, 09235</td>
</tr>
<tr>
<td>MS</td>
<td>0525</td>
<td>Steven D. Wix, 01734</td>
</tr>
<tr>
<td>MS 0525</td>
<td>Thomas V. Russo, 01734</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>MS 0525</td>
<td>Regina Schells, 01734</td>
<td></td>
</tr>
<tr>
<td>MS 0525</td>
<td>Carolyn Bogdan, 01734</td>
<td></td>
</tr>
<tr>
<td>MS 0525</td>
<td>Mike Deveney, 01734</td>
<td></td>
</tr>
<tr>
<td>MS 0525</td>
<td>Raymond B. Heath, 01734</td>
<td></td>
</tr>
<tr>
<td>MS 0525</td>
<td>Ronald Sikorksi, 01734</td>
<td></td>
</tr>
<tr>
<td>MS 0525</td>
<td>Albert Nunez, 01734</td>
<td></td>
</tr>
<tr>
<td>MS 1081</td>
<td>Paul E. Dodd, 01762</td>
<td></td>
</tr>
<tr>
<td>MS 0660</td>
<td>Roger F. Billau, 09519</td>
<td></td>
</tr>
<tr>
<td>MS 0874</td>
<td>Robert Brocato, 01751</td>
<td></td>
</tr>
<tr>
<td>MS 1081</td>
<td>Charles E. Hembree, 01739</td>
<td></td>
</tr>
<tr>
<td>MS 0889</td>
<td>Neil R. Sorenson, 01832</td>
<td></td>
</tr>
<tr>
<td>MS 0311</td>
<td>Greg Lyons, 02616</td>
<td></td>
</tr>
<tr>
<td>MS 0311</td>
<td>Martin Stevenson, 02616</td>
<td></td>
</tr>
<tr>
<td>MS 0328</td>
<td>Fred Anderson, 02612</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>Perry Molley, 02331</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>Siviengxay Limary, 02331</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>John Dye, 02331</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>Barbara Wampler, 02331</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>Doug Weiss, 02333</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>Scott Holswade, 02333</td>
<td></td>
</tr>
<tr>
<td>MS 0537</td>
<td>George R. Laguna, 02333</td>
<td></td>
</tr>
<tr>
<td>MS 0405</td>
<td>Todd R. Jones, 12333</td>
<td></td>
</tr>
<tr>
<td>MS 0405</td>
<td>Thomas D. Brown, 12333</td>
<td></td>
</tr>
<tr>
<td>MS 0405</td>
<td>Donald C. Evans, 12333</td>
<td></td>
</tr>
<tr>
<td>MS 0405</td>
<td>Matthew T. Kerschen, 12333</td>
<td></td>
</tr>
<tr>
<td>MS 9101</td>
<td>Rex Eastin, 08232</td>
<td></td>
</tr>
<tr>
<td>MS 9101</td>
<td>Seung Choi, 08235</td>
<td></td>
</tr>
<tr>
<td>MS 9409</td>
<td>William P. Ballard, 08730</td>
<td></td>
</tr>
<tr>
<td>MS 9202</td>
<td>Kathryn R. Hughes, 08205</td>
<td></td>
</tr>
<tr>
<td>MS 9202</td>
<td>Rene L. Bierbaum, 08205</td>
<td></td>
</tr>
<tr>
<td>MS 9202</td>
<td>Kenneth D. Marx, 08205</td>
<td></td>
</tr>
<tr>
<td>MS 9202</td>
<td>Stephen L. Brandon, 08205</td>
<td></td>
</tr>
<tr>
<td>MS 9202</td>
<td>Jason Dimkoff, 08205</td>
<td></td>
</tr>
<tr>
<td>MS 9202</td>
<td>Brian E. Owens, 08205</td>
<td></td>
</tr>
</tbody>
</table>