

# **Sentaurus™ Device User Guide**

---

Version T-2022.03, March 2022

**SYNOPSYS®**

# **Copyright and Proprietary Information Notice**

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

## **Destination Control Statement**

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

## **Disclaimer**

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

## **Trademarks**

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

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

## **Free and Open-Source Licensing Notices**

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

## **Third-Party Links**

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

[www.synopsys.com](http://www.synopsys.com)

# Contents

---

Conventions . . . . .	47
Customer Support . . . . .	48
Acknowledgments . . . . .	49

---

## Part I: Getting Started

---

1. Introduction to Sentaurus Device . . . . .	51
Sentaurus Device Functionality . . . . .	51
Creating and Meshing Device Structures . . . . .	53
Tool Flow . . . . .	53
Generating Input Files for Sentaurus Device . . . . .	54
Starting Sentaurus Device . . . . .	55
From the Command Line . . . . .	55
From Sentaurus Workbench . . . . .	56
TCAD Sentaurus Tutorial: Simulation Projects . . . . .	56
2. Specifying Physical Devices . . . . .	57
Reading a Structure . . . . .	57
Cylindrical Coordinate System . . . . .	58
Abrupt and Graded Heterojunctions . . . . .	59
Specifying Doping Species . . . . .	60
Specifying Materials . . . . .	62
User-Defined Materials . . . . .	63
Mole-Fraction Materials . . . . .	64
Mole-Fraction Specification . . . . .	65
Physical Models and the Hierarchy of Their Specification . . . . .	66
Region-Specific and Material-Specific Models . . . . .	67
Interface-Specific Models . . . . .	68
Electrode-Specific Models . . . . .	69

## Contents

---

Physical Model Parameters . . . . .	70
Search Strategy for Parameter Files . . . . .	71
Parameters for Composition-Dependent Materials . . . . .	72
Ternary Semiconductor Composition . . . . .	74
Quaternary Semiconductor Composition . . . . .	76
Default Model Parameters for Compound Semiconductors . . . . .	77
Combining Parameter Specifications . . . . .	78
Materialwise Parameters . . . . .	79
Regionwise Parameters . . . . .	79
Material Interface–Wise Parameters . . . . .	80
Region Interface–Wise Parameters . . . . .	80
Electrode–Wise Parameters . . . . .	80
Generating a Copy of Parameter File . . . . .	80
Undefined Physical Models . . . . .	82
Default Parameters . . . . .	84
Named Parameter Sets . . . . .	85
Auto-Orientation Framework . . . . .	86
Changing Orientations Used With Auto-Orientation . . . . .	87
Auto-Orientation Smoothing . . . . .	87
References . . . . .	88
<b>3. Mixed-Mode Simulations . . . . .</b>	<b>90</b>
Overview of Mixed-Mode Simulations . . . . .	90
Compact Models . . . . .	91
Hierarchical Description of Compact Models . . . . .	91
User-Defined Compact Models . . . . .	94
PrimeSim HSPICE Netlist Files . . . . .	95
Structure of Netlist Files . . . . .	95
Comments . . . . .	95
Continuation Lines . . . . .	96
The .INCLUDE Statement . . . . .	96
Numeric Constants . . . . .	96
Parameters and Expressions . . . . .	97
Subcircuits . . . . .	97
The .MODEL Statement . . . . .	98
Elements . . . . .	99
Netlist Commands . . . . .	100
Specifying Physical Devices in Netlists . . . . .	100
SPICE Circuit Files . . . . .	101
Example . . . . .	102

## Contents

---

Command File for Mixed-Mode Simulations . . . . .	103
System Section . . . . .	103
Physical Devices . . . . .	105
Circuit Devices . . . . .	105
Electrical and Thermal Netlists . . . . .	106
Initializing Nodes . . . . .	108
Plotting Quantities . . . . .	109
Modifying Plot Output . . . . .	109
Device Section . . . . .	110
File Section . . . . .	110
Math Section . . . . .	111
Working With Mixed-Mode Simulations . . . . .	112
From Single-Device to Multidevice Command Files . . . . .	112
File-Naming Conventions for Mixed Mode . . . . .	113
<b>4. Performing Numeric Experiments . . . . .</b>	<b>115</b>
Specifying Electrical Boundary Conditions . . . . .	115
Changing Boundary Condition Type During Simulation . . . . .	116
Mixed-Mode Electrical Boundary Conditions . . . . .	117
Specifying Thermal Boundary Conditions . . . . .	118
Break Criteria: Conditionally Stopping the Simulation . . . . .	119
Global Contact Break Criteria . . . . .	120
Global Device Break Criteria . . . . .	120
Sweep-Specific Break Criteria . . . . .	121
Mixed-Mode Break Criteria . . . . .	122
Quasistationary Ramps . . . . .	122
Ramping Boundary Conditions . . . . .	123
Ramping Quasi-Fermi Potentials in Doping Wells . . . . .	124
Ramping Physical Parameter Values . . . . .	126
Quasistationary in Mixed Mode . . . . .	129
Saving and Plotting During a Quasistationary . . . . .	130
Extrapolation . . . . .	130
Inside a Quasistationary Command . . . . .	131
Between Quasistationary Commands . . . . .	131
Parameters of the Extrapolate Statement . . . . .	133
Continuation Command . . . . .	135
Continuation Control Parameters . . . . .	136
Continuation in Mixed Mode . . . . .	137

## Contents

Transient Command . . . . .	138
Numeric Control of Transient Analysis . . . . .	139
Time-Stepping . . . . .	140
Ramping Physical Parameter Values . . . . .	142
Extrapolation . . . . .	143
Inside a Transient Command . . . . .	143
Between Quasistationary Commands . . . . .	143
Transient Ramps . . . . .	144
Large-Signal Cyclic Analysis . . . . .	145
Description of Method . . . . .	145
Using Cyclic Analysis . . . . .	148
Small-Signal AC Analysis . . . . .	149
AC Analysis in Mixed-Mode Simulations . . . . .	149
Example . . . . .	150
AC Analysis in Single-Device Mode . . . . .	152
Optical AC Analysis . . . . .	153
Harmonic Balance . . . . .	153
Modes of Harmonic Balance Analysis . . . . .	154
MDFT Mode . . . . .	154
SDFT Mode . . . . .	155
Performing Harmonic Balance Analysis . . . . .	155
Solve Spectrum . . . . .	156
Convergence Parameters . . . . .	156
Additional Remarks . . . . .	157
Harmonic Balance Analysis Output . . . . .	158
Circuit Currents and Voltages . . . . .	158
Solution Variables . . . . .	159
Additional Plot Variables . . . . .	159
Output of Individual HBCoupled Statements . . . . .	160
Application Notes . . . . .	160
References . . . . .	160
<hr/>	
5. Simulation Results . . . . .	162
Current File . . . . .	162
When to Write to the Current File . . . . .	162
Example: CurrentPlot Statements . . . . .	164
NewCurrentPrefix Statement . . . . .	164
Tracking Additional Data in the Current File . . . . .	165

## Contents

CurrentPlot Section . . . . .	166
Example: Mixed Mode . . . . .	168
Example: Advanced Options . . . . .	168
Example: Plotting Parameter Values . . . . .	169
CurrentPlot Options . . . . .	169
Tcl Formulas. . . . .	170
Dataset Option . . . . .	171
Function Option . . . . .	172
Unit Option . . . . .	172
Init Option. . . . .	172
Formula Option. . . . .	172
Finish Option . . . . .	173
Operation Option . . . . .	173
Examples . . . . .	175
Device Plots. . . . .	177
What to Plot . . . . .	177
When to Plot. . . . .	178
Snapshots . . . . .	179
Interface Plots . . . . .	180
Plotting Results Along Carrier Paths . . . . .	181
Saddle Point Analysis in CIS Devices . . . . .	181
Log File . . . . .	183
Extraction File . . . . .	184
Extraction File Format . . . . .	184
Analysis Modes . . . . .	185
File Section . . . . .	186
Electrode Section. . . . .	186
Extraction Section . . . . .	187
Solve Section . . . . .	187
<hr/> 6. Numeric and Software-Related Issues. . . . .	189
Structure of Command File . . . . .	189
Inserting Files. . . . .	189
Solve Section: How the Simulation Proceeds . . . . .	190
Nonlinear Iterations . . . . .	191
Coupled Statement . . . . .	191
Convergence and Error Control . . . . .	193
Damped Newton Iterations. . . . .	195
Derivatives . . . . .	196

## Contents

Incomplete Newton Algorithm . . . . .	196
Additional Equations Available in Mixed Mode . . . . .	197
Selecting Individual Devices in Mixed Mode . . . . .	198
Relaxed Newton Method . . . . .	198
Plugin Command . . . . .	200
Linear Solvers . . . . .	201
Nonlocal Meshes . . . . .	202
Specifying Nonlocal Meshes . . . . .	202
Visualizing Nonlocal Meshes . . . . .	203
Visualizing Data Defined on Nonlocal Meshes . . . . .	204
Constructing Nonlocal Meshes . . . . .	204
Specification Using Barrier . . . . .	205
Specification Using a Reference Surface . . . . .	205
Special Handling of the 1D Schrödinger Equation . . . . .	207
Special Handling of the Nonlocal Tunneling Model . . . . .	207
Unnamed Meshes . . . . .	208
Performance Suggestions . . . . .	208
Monitoring Convergence Behavior . . . . .	209
CNormPrint . . . . .	209
NewtonPlot . . . . .	209
Automatic Activation of CNormPrint and NewtonPlot . . . . .	210
Simulation Statistics for Plotting and Output . . . . .	211
Simulation Statistics in Current Plot Files . . . . .	211
Simulation Statistics in Design-of-Experiments Variables . . . . .	212
Save and Load Statements . . . . .	213
Tcl Command File . . . . .	214
Performing Device Simulations . . . . .	215
sdevice Command . . . . .	217
sdevice_init Command . . . . .	217
sdevice_solve Command . . . . .	218
sdevice_finish Command . . . . .	218
sdevice_parameters Command . . . . .	218
Extracting Parameters . . . . .	218
Available Inspect Tcl Commands . . . . .	219
Redirecting Output . . . . .	220
Known Restrictions . . . . .	220
Parallelization . . . . .	221
Extended Precision . . . . .	223

## Contents

---

System Command .....	225
References.....	226
<b>Part II: Physics in Sentaurus Device</b>	
<hr/>	
<b>7. Poisson Equation and Quasi-Fermi Potentials.....</b>	229
Electrostatic Potential .....	229
Dipole Layer.....	230
Equilibrium Solution .....	231
Quasi-Fermi Potential With Boltzmann Statistics.....	231
Fermi Statistics .....	233
Using Fermi Statistics .....	233
Initial Guess for Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells .....	234
Regionwise Specification of Initial Quasi-Fermi Potentials .....	235
Alternative Initial Guess for Quasi-Fermi Potentials From Linear Conduction Equation.....	235
Electrode Charge Calculation .....	237
<hr/>	
<b>8. Carrier Transport Equations in Semiconductors .....</b>	238
Introduction to Carrier Transport Models .....	238
Drift-Diffusion Model .....	239
Thermodynamic Model for Current Densities .....	240
Hydrodynamic Model for Current Densities .....	241
Numeric Parameters for Continuity Equation.....	241
Numeric Approaches for Contact Current Computation.....	242
Current Potential .....	242
References.....	243
<hr/>	
<b>9. Lattice and Carrier Temperature Equations .....</b>	245
Introduction to Temperature Equations .....	245
Uniform Self-Heating .....	246

## Contents

---

Using Uniform Self-Heating . . . . .	247
Default Model for Lattice Temperature . . . . .	248
Thermodynamic Model for Lattice Temperature . . . . .	249
Total Heat and Its Contributions . . . . .	249
Using the Thermodynamic Model . . . . .	250
Hydrodynamic Model for Temperatures . . . . .	252
Hydrodynamic Model Parameters . . . . .	254
Using the Hydrodynamic Model . . . . .	255
Numeric Parameters for Temperature Equations . . . . .	256
Validity Ranges for Lattice and Carrier Temperatures . . . . .	256
Scaling of Lattice Heat Generation . . . . .	256
References . . . . .	257
<b>10. Boundary Conditions . . . . .</b>	<b>258</b>
Electrical Boundary Conditions . . . . .	258
Ohmic Contacts . . . . .	258
Modified Ohmic Contacts . . . . .	259
Contacts on Insulators . . . . .	260
Schottky Contacts . . . . .	261
Fermi-Level Pinning at Schottky Contacts . . . . .	262
Barrier Lowering at Schottky Contacts . . . . .	265
Resistive Contacts . . . . .	270
Resistive Interfaces . . . . .	277
Boundaries Without Contacts . . . . .	279
Application Notes . . . . .	279
Floating Contacts . . . . .	280
Floating Metal Contacts . . . . .	280
Floating Semiconductor Contacts . . . . .	281
Thermal Boundary Conditions . . . . .	283
Boundary Conditions for Lattice Temperature . . . . .	283
Boundary Conditions for Carrier Temperatures . . . . .	284
Periodic Boundary Conditions . . . . .	284
Robin PBC Approach . . . . .	285
Mortar PBC Approach . . . . .	285
Specifying Periodic Boundary Conditions . . . . .	286
Specifying Robin Periodic Boundary Conditions . . . . .	286

## Contents

---

Specifying Mortar Periodic Boundary Conditions .....	287
Application Notes .....	287
Specialized Linear Solver for MPBC .....	288
Discontinuous Interfaces .....	288
Representation of Physical Quantities Across Interfaces .....	288
Interface Conditions at Discontinuous Interfaces .....	289
Critical Points .....	289
References .....	289
<b>11. Transport Equations in Metals, Organic Materials, and Disordered Media</b> .....	291
Singlet Exciton Equation .....	291
Boundary and Continuity Conditions for Singlet Exciton Equation .....	292
Using the Singlet Exciton Equation .....	293
Transport in Metals .....	295
Electric Boundary Conditions for Metals .....	296
Metal Workfunction .....	298
Metal Workfunction Randomization .....	299
Temperature in Metals .....	300
Conductive Insulators .....	300
<b>12. Semiconductor Band Structure</b> .....	304
Intrinsic Density .....	304
Band Gap and Electron Affinity .....	305
Selecting the Bandgap Model .....	305
Bandgap and Electron-Affinity Models .....	305
Bandgap Narrowing for Bennett–Wilson Model .....	306
Bandgap Narrowing for Slotboom Model .....	307
Bandgap Narrowing for del Alamo Model .....	307
Bandgap Narrowing for Jain–Roulston Model .....	307
Table Specification of Bandgap Narrowing .....	309
Schenk Bandgap Narrowing Model .....	310
Bandgap Narrowing With Incomplete Ionization .....	314
Bandgap Narrowing With Fermi Statistics .....	315
Band Gap and Parameters of Non-Three-Dimensional Density-of-States .....	316
Bandgap Parameters .....	317
Effective Masses and Effective Density-of-States .....	319
Electron Effective Mass and Density-of-States .....	319

## Contents

---

Formula=1 .....	319
Formula=2 .....	320
Electron Effective Mass and Conduction Band DOS Parameters .....	320
Hole Effective Mass and Density-of-States .....	321
Formula=1 .....	321
Formula=2 .....	321
Hole Effective Mass and Valence Band DOS Parameters.....	321
Gaussian Density-of-States for Organic Semiconductors .....	322
Band Tails .....	325
Multivalley Band Structure.....	327
Nonparabolic Band Structure.....	328
Bandgap Widening.....	329
Monte Carlo Density-of-States.....	330
Non-Three-Dimensional Density-of-States .....	330
Graphene Nanoribbon Density-of-States.....	331
Carbon Nanotube Density-of-States .....	331
Using Multivalley Band Structure .....	331
References.....	337
<b>13. Incomplete Ionization .....</b>	<b>339</b>
Considering Incomplete Ionization.....	339
Using the Incomplete Ionization Model .....	339
Incomplete Ionization Model .....	340
Physical Model Parameters .....	343
Multiple Lattice Sites .....	345
References.....	347
<b>14. Quantization.....</b>	<b>348</b>
Modeling Quantization Effects.....	348
The van Dort Model.....	349
Using the van Dort Model.....	350
1D Schrödinger Equation.....	351
Nonlocal Mesh for the 1D Schrödinger Equation.....	351
Using the 1D Schrödinger Equation.....	352
Parameters of the 1D Schrödinger Equation.....	352

## Contents

Explicitly Specifying Ladders . . . . .	353
Automatically Extracting Ladder Parameters . . . . .	353
Visualizing the Solutions . . . . .	355
1D Schrödinger Equation . . . . .	355
Notes on the Use of the 1D Schrödinger Equation . . . . .	357
External 2D Schrödinger Solver . . . . .	357
Notes on the Use of External 2D Schrödinger Solvers . . . . .	359
External Boltzmann Solver . . . . .	359
Density Gradient Model . . . . .	362
Using the Density Gradient Model . . . . .	363
Named Parameter Sets for the Density Gradient Model . . . . .	367
Auto-Orientation for the Density Gradient Model . . . . .	367
Notes on the Use of the Density Gradient Model . . . . .	367
Modified Local-Density Approximation Model . . . . .	370
Interface Orientation and Stress Dependencies . . . . .	370
Heterojunctions . . . . .	372
Nonparabolic Bands and Geometric Quantization . . . . .	372
Using the MLDA Model . . . . .	373
Notes on the Use of the MLDA Model . . . . .	377
Quantum-Well Quantization Model . . . . .	378
Extracting Layer Thickness . . . . .	379
Combining LayerThickness Command and ThinLayer Subcommand . . . . .	380
Geometric Parameters of LayerThickness Command . . . . .	381
Thickness Extraction . . . . .	382
References . . . . .	384
<hr/>	
15. Mobility . . . . .	385
Introduction to Mobility Models . . . . .	385
Combining Mobility Models . . . . .	385
Mobility due to Phonon Scattering . . . . .	386
Doping-Dependent Mobility Degradation . . . . .	387
Using Doping-Dependent Mobility . . . . .	387
Using More Than One Doping-Dependent Mobility Model . . . . .	388
Masetti Model . . . . .	388
Arora Model . . . . .	389
University of Bologna Bulk Mobility Model . . . . .	390

## Contents

The pmi_msc_mobility Model . . . . .	393
PMIs for Bulk Mobility . . . . .	394
Low-Field Ballistic Mobility Models . . . . .	394
Simple Channel Length–Dependent Ballistic Mobility Model . . . . .	394
Injection Velocity–Dependent Ballistic Mobility Model . . . . .	395
Carrier–Carrier Scattering . . . . .	396
Using Carrier–Carrier Scattering . . . . .	396
Conwell–Weisskopf Model . . . . .	397
Brooks–Herring Model . . . . .	397
Physical Model Parameters . . . . .	397
Philips Unified Mobility Model . . . . .	398
Using the Philips Unified Mobility Model . . . . .	398
Using an Alternative Philips Unified Mobility Model . . . . .	399
Description of the Philips Unified Mobility Model . . . . .	399
Screening Parameter . . . . .	401
Philips Model Parameters . . . . .	401
Mobility Degradation at Interfaces . . . . .	403
Using Mobility Degradation at Interfaces . . . . .	403
Enhanced Lombardi Model . . . . .	404
Stress Factors for Mobility Components . . . . .	406
Named Parameter Sets for the Lombardi Model . . . . .	407
Auto-Orientation for the Lombardi Model . . . . .	407
Inversion and Accumulation Layer Mobility Model . . . . .	407
Coulomb Scattering . . . . .	408
Phonon Scattering . . . . .	410
Surface Roughness Scattering . . . . .	410
Parameters . . . . .	411
Stress Factors for Mobility Components . . . . .	414
Using the Inversion and Accumulation Layer Mobility Model . . . . .	415
Named Parameter Sets for the IALMob Model . . . . .	415
Auto-Orientation for the IALMob Model . . . . .	416
University of Bologna Surface Mobility Model . . . . .	416
Mobility Degradation Components due to Coulomb Scattering . . . . .	419
Stress Factors for Mobility Components . . . . .	421
Using Mobility Degradation Components . . . . .	421
Remote Coulomb Scattering Model . . . . .	423
Stress Factors for Mobility Components . . . . .	424
Remote Phonon Scattering Model . . . . .	425
Stress Factors for Mobility Components . . . . .	426
Computing Transverse Field . . . . .	426

## Contents

Normal to the Interface.....	426
Normal to the Current Flow .....	427
Field Correction on Interface .....	427
Thin-Layer Mobility Model .....	427
Using the Thin-Layer Mobility Model .....	429
Physical Parameters .....	429
Stress Factors for Mobility Components .....	432
Auto-Orientation and Named Parameter Sets .....	432
Geometric Parameters .....	433
Carbon Nanotube Mobility Model.....	433
Using the CNT Mobility Model .....	434
Low-Temperature Phonon Exponent Correction for Mobility Models.....	435
Constant Mobility Model.....	436
Philips Unified Mobility Model .....	436
Enhanced Lombardi Mobility Model.....	437
Inversion and Accumulation Layer Mobility Model.....	438
High-Field Saturation Models.....	438
Using High-Field Saturation Models.....	439
Named Parameter Sets for the High-Field Saturation Models .....	440
Auto-Orientation for the High-Field Saturation Models .....	440
Extended Canali Model .....	440
Transferred Electron Model .....	442
Transferred Electron Model 2.....	442
Multivalley Transferred Electron Mobility .....	444
Basic Model .....	444
Meinerzhagen–Engl Model .....	445
Physical Model Interfaces .....	445
Lucent Model .....	446
Velocity Saturation Models.....	446
Selecting Velocity Saturation Models .....	446
Driving Force Models .....	447
Electric Field Parallel to the Current.....	447
Gradient of Quasi-Fermi Potential .....	448
Electric Field Parallel to the Interface .....	448
Hydrodynamic Driving Force .....	450
Electric Field.....	450
Interpolation of Driving Forces to Zero Field .....	450
Interpolation of the GradQuasiFermi Driving Force .....	451
Interpolation of the Eparallel Driving Force .....	452
Field Correction Close to Interfaces .....	452

## Contents

Non-Einstein Diffusivity .....	453
Band Tail Mobility .....	454
High-Field Saturation Mobility Scaling .....	455
High-Field Mobility in Disordered Materials .....	455
Poole–Frenkel Mobility .....	455
Variable Range Hopping Transport Mobility .....	456
Ballistic Mobility Model .....	458
Channel Length–Dependent Model .....	458
Kinetic Velocity Model .....	459
Fermi–Dirac Statistics .....	460
Frensley Rule .....	460
Using the Ballistic Mobility Model .....	460
Using the PMI Fitting Parameter in the Ballistic Mobility Model .....	461
General Model for Mobility Degradation by Traps in the Bulk and at Semiconductor Interfaces .....	462
Incomplete Ionization–Dependent Mobility Models .....	463
Mobility Averaging .....	464
Mobility Doping File .....	465
Effective Mobility .....	465
EffectiveMobility PMI Methods .....	467
Using the EffectiveMobility PMI .....	468
References .....	469
<hr/>	
<b>16. Generation–Recombination .....</b>	<b>473</b>
Shockley–Read–Hall Recombination .....	473
Using SRH Recombination .....	474
SRH Doping Dependence .....	474
Lifetime Profiles From Files .....	475
Improved Nakagawa Model .....	475
SRH Temperature Dependence .....	476
SRH Doping- and Temperature-Dependent Parameters .....	477
SRH Field Enhancement .....	477
Using Field Enhancement .....	478
Schenk Trap-Assisted Tunneling Model .....	479
Hurkx Trap-Assisted Tunneling Model .....	480
Density Correction for Schenk and Hurkx Trap-Assisted Tunneling Models .....	481

## Contents

Dynamic Nonlocal Path Trap-Assisted Tunneling . . . . .	481
Recombination Rate . . . . .	482
Using the Dynamic Nonlocal Path Trap-Assisted Tunneling Model . . . . .	483
Trap-Assisted Auger Recombination Model . . . . .	485
Surface SRH Recombination Model . . . . .	486
Coupled Defect Level Recombination . . . . .	487
Using Coupled Defect Level Recombination . . . . .	488
Radiative Recombination Model . . . . .	488
Using the Radiative Recombination Model . . . . .	489
Auger Recombination Model . . . . .	489
Using the Auger Recombination Model . . . . .	490
Intrinsic Recombination Model for Silicon . . . . .	490
Constant Carrier Generation Model . . . . .	494
Avalanche Generation . . . . .	494
Using Avalanche Generation . . . . .	495
van Overstraeten – de Man Model . . . . .	497
Okuto–Crowell Model . . . . .	499
Lackner Model . . . . .	500
University of Bologna Impact Ionization Model . . . . .	501
New University of Bologna Impact Ionization Model . . . . .	502
Hatakeyama Avalanche Model . . . . .	505
Driving Force of Hatakeyama Avalanche Model . . . . .	506
Default Anisotropic Coordinate System . . . . .	507
Specification of Anisotropic Direction . . . . .	507
Driving Force . . . . .	509
Interpolation of Avalanche Driving Forces . . . . .	509
Avalanche Generation With Hydrodynamic Transport . . . . .	510
Approximate Breakdown Analysis . . . . .	511
Using Breakdown Analysis . . . . .	512
Stopping Criteria for Breakdown Simulations . . . . .	515
Visualizing Breakdown Paths . . . . .	515
Typical Command File of Sentaurus Device . . . . .	517
Approximate Breakdown Analysis With Carriers . . . . .	517
Using GradQuasiFermi as a Driving Force . . . . .	518
Avalanche Breakdown Probability . . . . .	518
Using Avalanche Breakdown Probability . . . . .	519
High-Field Entrance Position and Time . . . . .	520

## Contents

Using High-Field Entrance Position and Time .....	520
Visualization in TDR Files .....	522
Save Volume Dataset in Existing TDR File .....	522
Save Carrier Paths as 1D Datasets in Separate TDR File.....	522
Band-to-Band Tunneling Models .....	523
Using Band-to-Band Tunneling .....	524
Tunneling Near Interfaces and Equilibrium Regions .....	525
Schenk Band-to-Band Tunneling Model.....	526
Schenk Density Correction.....	527
Hurkx Band-to-Band Tunneling Model.....	527
Modified Hurkx Band-to-Band Tunneling Model.....	528
Simple Band-to-Band Tunneling Models .....	529
Dynamic Nonlocal Path Band-to-Band Tunneling Model.....	529
Band-to-Band Generation Rate .....	531
Using the Nonlocal Path Band-to-Band Tunneling Model .....	533
Handling Derivatives .....	535
Tunneling Parameters From the Physical Model Interface.....	536
Energy Shift due to Geometric Confinement .....	536
Postprocessing Mode.....	537
Frozen Tunneling Direction .....	538
Visualizing Nonlocal Band-to-Band Generation Rate.....	538
Bimolecular Recombination Model .....	538
Using the Bimolecular Recombination Model .....	539
Exciton Dissociation Model .....	539
Using the Exciton Dissociation Model .....	540
References.....	540
<hr/>	
<b>17. Traps and Fixed Charges .....</b>	<b>543</b>
Introduction to Traps .....	543
Syntax for Traps.....	543
Trap Types .....	544
Energetic and Spatial Distribution of Traps .....	544
Specifying Single Traps .....	548
Trap Randomization.....	549
Explicit Trap Occupation .....	550
Options to Include Traps in Doping .....	552
Trap Examples.....	552
Trap Models.....	553

## Contents

Trap Occupation Dynamics . . . . .	553
Local Capture and Emission Rates . . . . .	555
J-Model for Cross Sections . . . . .	556
Hurkx Model for Cross Sections . . . . .	557
Poole–Frenkel Model for Cross Sections . . . . .	557
Makram-Ebeid-Lannoo Model . . . . .	559
Local Capture and Emission Rates From PMI . . . . .	561
Trap-to-Trap Tunneling (DiscreteTrapT2T) . . . . .	561
Mode Oscillator Model . . . . .	567
Inelastic Phonon Model . . . . .	568
Trap-to-Trap Recombination . . . . .	570
Trap-to-Trap Flux of InelasticPhonon Model . . . . .	572
Coulomb Potential–Induced Barrier Lowering for Inelastic Phonon Trap-to-Trap Tunneling . . . . .	572
Nonlocal Tunneling for Traps . . . . .	576
Electron Capture Rate for the Phonon-Assisted Transition . . . . .	576
Electron Capture Rate for the Elastic Transition . . . . .	577
Numeric Parameters for Traps . . . . .	578
Visualizing Traps . . . . .	578
Insulator Fixed Charges . . . . .	582
References . . . . .	582
<hr/>	
18. Phase and State Transitions . . . . .	584
Multistate Configurations and Their Dynamic . . . . .	584
Specifying Multistate Configurations . . . . .	586
Multistate Configurations on Interfaces . . . . .	587
Additional Remarks . . . . .	587
Transition Models . . . . .	588
The pmi_ce_msc Model . . . . .	588
States . . . . .	589
Transitions . . . . .	589
Model Parameters . . . . .	592
Interaction of Multistate Configurations With Transport . . . . .	594
Apparent Band-Edge Shift . . . . .	595
The pmi_msc_abes Model . . . . .	595
Thermal Conductivity, Heat Capacity, and Mobility . . . . .	596
Manipulating MSCs During Solve . . . . .	597
Explicit State Occupations . . . . .	597

## Contents

---

Manipulating Transition Dynamics . . . . .	598
Example: Two-State Phase-Change Memory Model . . . . .	598
References . . . . .	599
<b>19. Degradation . . . . .</b>	<b>600</b>
Overview of Degradation Models . . . . .	600
Trap Degradation Model . . . . .	601
Trap Formation Kinetics . . . . .	601
Power Law and Kinetic Equation . . . . .	601
Si-H Density–Dependent Activation Energy . . . . .	602
Diffusion of Hydrogen in Oxide . . . . .	603
Model Equations and Syntax . . . . .	603
Reaction Enhancement Factors . . . . .	605
Using the Trap Degradation Model . . . . .	606
Device Lifetime and Simulation . . . . .	607
Degradation in Insulators . . . . .	609
Fluence Model . . . . .	611
MSC–Hydrogen Transport Degradation Model . . . . .	611
Hydrogen Transport . . . . .	612
Boundary Conditions . . . . .	613
Reactions Between Mobile Elements . . . . .	614
Reactions With Multistate Configurations . . . . .	616
The CEModel_Depassivation Model . . . . .	617
Using MSC–Hydrogen Transport Degradation Model . . . . .	619
Changing Charge and Hydrogen Composition of Species . . . . .	621
Two-Stage NBTI Degradation Model . . . . .	621
Formulation . . . . .	622
Using Two-Stage NBTI Model . . . . .	624
Extended Nonradiative Multiphonon Model . . . . .	625
eNMP Model Description . . . . .	626
Using the eNMP Model . . . . .	628
eNMP Quantities Available for Plotting . . . . .	629
eNMP Model Parameters . . . . .	630
eNMP Transition Rates PMI Model . . . . .	631
Hot-Carrier Stress Degradation Model . . . . .	631
Model Description . . . . .	631
Single-Particle and Multiple-Particle Interface-Trap Densities . . . . .	632
Field-Enhanced Thermal Degradation . . . . .	633

## Contents

---

Carrier Distribution Function.....	634
Bond Dispersion.....	635
Using the HCS Degradation Model .....	636
Activated Barrier Double Well Thermionic Model .....	638
Model Description .....	638
Using the ABDWT Model .....	639
Transient Trap Occupancy Model .....	641
Model Description .....	641
Using the Transient Trap Occupancy Model .....	641
References.....	643
<b>20. Organic Devices.....</b>	645
Introduction to Organic Device Simulation.....	645
References.....	647
<b>21. Optical Generation.....</b>	648
Overview of Optical Generation.....	648
Specifying the Type of Optical Generation Computation .....	649
Optical Generation From Monochromatic Source .....	651
Illumination Spectrum. ....	651
Multidimensional Illumination Spectra .....	652
Enhanced Spectrum Control .....	653
Loading and Saving Optical Generation From and to Files .....	656
Constant Optical Generation .....	657
Quantum Yield Models.....	658
Optical Absorption Heat .....	660
Specifying Time Dependency for Transient Simulations .....	661
Optical Turning Points .....	667
Solving the Optical Problem.....	668
Specifying the Optical Solver.....	669
Transfer Matrix Method .....	669
Finite-Difference Time-Domain Method .....	670
Raytracing .....	671
Beam Propagation Method. ....	673
Loading Solution of Optical Problem From File .....	673
Optical Beam Absorption Method.....	674
Composite Method .....	674

## Contents

Setting the Excitation Parameters .....	674
Illumination Window .....	676
Spatial Intensity Function Excitation.....	684
Choosing Refractive Index Model .....	686
Extracting the Layer Stack.....	686
Controlling Computation of Optical Problem in Solve Section .....	688
Parameter Ramping.....	689
Accurate Absorbed Photon Density for 1D Optical Solvers.....	691
Complex Refractive Index Model.....	694
Physical Model.....	694
Wavelength Dependency .....	695
Temperature Dependency .....	695
Carrier Dependency .....	695
Gain Dependency.....	697
Using Complex Refractive Index .....	697
Raytracing .....	702
Raytracer .....	702
Ray Photon Absorption and Optical Generation .....	704
Using the Raytracer .....	705
Terminating Raytracing .....	707
Monte Carlo Raytracing .....	707
Multithreading for Raytracer.....	708
Compact Memory Model for Raytracer .....	708
Window of Starting Rays .....	709
User-Defined Window of Rays.....	709
Distribution Window of Rays .....	711
Cylindrical Coordinates for Raytracing.....	712
Boundary Condition for Raytracing .....	713
Fresnel Boundary Condition.....	714
Constant Reflectivity and Transmittivity Boundary Condition.....	715
Raytrace PMI Boundary Condition.....	716
Thin-Layer-Stack Boundary Condition .....	717
TMM Optical Generation in Raytracer .....	718
Diffuse Surface Boundary Condition .....	720
Periodic Boundary Condition .....	722
Virtual Regions in Raytracer .....	723
External Material in Raytracer .....	723
Additional Options for Raytracing.....	724
Redistributing Power of Stopped Rays .....	724
Weighted Interpolation for Raytrace Optical Generation .....	725

## Contents

Visualizing Raytracing .....	726
Computing Optical Intensity.....	726
Reporting Various Powers in Raytracing .....	727
Plotting Interface Flux .....	728
Far Field and Sensors for Raytracing .....	730
Dual-Grid Setup for Raytracing .....	732
Transfer Matrix Method .....	734
Physical Model.....	734
Current Plot Quantities.....	736
Rough Surface Scattering .....	737
Using Transfer Matrix Method .....	739
Using Scattering Solver .....	741
Loading Solution of Optical Problem From Files .....	749
Importing 1D Profiles Into Higher-Dimensional Grids .....	752
Ramping Profile Index .....	752
Optical Beam Absorption Method .....	753
Physical Model.....	754
Using Optical Beam Absorption Method .....	755
Beam Propagation Method .....	756
Physical Model.....	756
Bidirectional BPM.....	757
Boundary Conditions .....	757
Using Beam Propagation Method .....	758
General.....	758
Bidirectional BPM.....	759
Excitation .....	760
Boundary .....	763
Ramping Input Parameters .....	763
Visualizing Results on Native Tensor Grid .....	764
Composite Method.....	765
Using the Composite Method.....	765
Controlling Interpolation When Loading Optical Generation Profiles.....	768
Optical AC Analysis .....	770
References.....	771
<hr/>	
22. Radiation.....	772
Carrier Generation by Gamma Radiation .....	772

## Contents

---

Yield Function . . . . .	773
Carrier Generation by Alpha Particles . . . . .	773
Using the Alpha Particle Model . . . . .	774
Carrier Generation by Heavy Ions . . . . .	775
Using the Heavy Ion Model . . . . .	776
Examples: Heavy Ions . . . . .	778
Example 1 . . . . .	778
Example 2 . . . . .	779
Example 3 . . . . .	779
References . . . . .	780
<b>23. Noise, Fluctuations, and Sensitivity . . . . .</b>	<b>781</b>
Using the Impedance Field Method . . . . .	781
Specifying Variations . . . . .	782
Specifying the Solver . . . . .	783
Analysis at Frequency Zero . . . . .	783
Output of Results . . . . .	784
Noise Sources . . . . .	787
Common Options . . . . .	787
Diffusion Noise . . . . .	787
Equivalent Monopolar Generation–Recombination Noise . . . . .	788
Bulk Flicker Noise . . . . .	788
Trapping Noise . . . . .	789
Random Dopant Fluctuations . . . . .	790
Random Geometric Fluctuations . . . . .	790
Random Trap Concentration Fluctuations . . . . .	793
Random Workfunction Fluctuations . . . . .	793
Random Band Edge Fluctuations . . . . .	794
Random Metal Conductivity Fluctuations . . . . .	795
Random Dielectric Constant Fluctuations . . . . .	796
Noise From SPICE Circuit Elements . . . . .	796
Statistical Impedance Field Method . . . . .	797
Options Common to sIFM Variations . . . . .	798
Spatial Correlations and Random Fields . . . . .	799
Doping Variations . . . . .	801
Trap Concentration Variations . . . . .	802
Workfunction Variations . . . . .	803

## Contents

---

Geometric Variations . . . . .	804
Band Edge Variations . . . . .	805
Metal Conductivity Variations . . . . .	806
Dielectric Constant Variations . . . . .	807
Doping Profile Variations . . . . .	807
Deterministic Variations . . . . .	808
Deterministic Doping Variations . . . . .	809
Deterministic Geometric Variations . . . . .	810
Parameter Variations . . . . .	811
IFM Section . . . . .	812
Impedance Field Method . . . . .	814
Noise Output Data . . . . .	815
References . . . . .	818
<hr/>	
<b>24. Tunneling . . . . .</b>	<b>819</b>
Overview of Tunneling Models . . . . .	819
Fowler–Nordheim Tunneling . . . . .	820
Using Fowler–Nordheim . . . . .	820
Fowler–Nordheim Model . . . . .	821
Fowler–Nordheim Parameters . . . . .	822
Direct Tunneling . . . . .	822
Using Direct Tunneling . . . . .	823
Direct Tunneling Model . . . . .	823
Image Force Effect . . . . .	824
Direct Tunneling Parameters . . . . .	825
Nonlocal Tunneling at Interfaces, Contacts, and Junctions . . . . .	826
Defining Nonlocal Meshes . . . . .	826
Specifying Nonlocal Tunneling Model . . . . .	828
Nonlocal Tunneling Parameters . . . . .	830
Visualizing Nonlocal Tunneling . . . . .	832
Physics of Nonlocal Tunneling Model . . . . .	832
WKB Tunneling Probability . . . . .	833
Schrödinger Equation–Based Tunneling Probability . . . . .	836
Density Gradient Quantization Correction . . . . .	837
Multivalley Band Structure and Geometric Quantization . . . . .	837
Nonlocal Tunneling Current . . . . .	838
Band-to-Band Contributions to Nonlocal Tunneling Current . . . . .	840

## Contents

---

Carrier Heating . . . . .	841
References . . . . .	842
<b>25. Hot-Carrier Injection . . . . .</b>	844
Overview of Hot-Carrier Injection Models . . . . .	844
Destination of Injected Current . . . . .	845
Injection Barrier and Image Potential . . . . .	847
Effective Field . . . . .	848
Classical Lucky Electron Injection . . . . .	849
Fiegna Hot-Carrier Injection . . . . .	850
SHE Distribution Hot-Carrier Injection . . . . .	852
Spherical Harmonics Expansion Method . . . . .	853
Using Spherical Harmonics Expansion Method . . . . .	857
Visualizing Spherical Harmonics Expansion Method . . . . .	866
Carrier Injection With Explicitly Evaluated Boundary Conditions for Continuity Equations . . . . .	867
References . . . . .	869
<b>26. Heterostructure Device Simulation . . . . .</b>	870
Thermionic Emission Current . . . . .	870
Using Thermionic Emission Current . . . . .	870
Thermionic Emission Model . . . . .	871
Thermionic Emission Model With Fermi Statistics . . . . .	872
Gaussian Transport Across Organic Heterointerfaces . . . . .	873
Using Gaussian Transport at Organic Heterointerfaces . . . . .	873
Gaussian Transport at Organic Heterointerface Model . . . . .	874
References . . . . .	874
<b>27. Energy-Dependent Parameters . . . . .</b>	875
Overview . . . . .	875
Energy-Dependent Energy Relaxation Time . . . . .	876
Spline Interpolation . . . . .	877
Energy-Dependent Mobility . . . . .	878
Spline Interpolation . . . . .	879

---

## Contents

---

Energy-Dependent Peltier Coefficient .....	880
Spline Interpolation .....	882
<b>28. Anisotropic Properties .....</b>	<b>883</b>
Anisotropic Properties of Semiconductor Devices.....	883
Anisotropic Approximations .....	884
TensorGridAniso .....	884
AnisoSG .....	885
StressSG .....	885
AverageAniso .....	886
Crystal and Simulation Coordinate Systems .....	886
Anisotropy Definition and Specification .....	887
Anisotropic Direction in 3D Cases .....	887
Anisotropic Direction in 2D Cases .....	888
Aniso Subsection With Direction Definition .....	888
Aniso Subsection Without Direction Definition .....	890
Aniso(direction) Versus LatticeParameters .....	892
Orthogonal Matrix From Eigenvectors Q .....	893
Anisotropic Mobility .....	893
Anisotropy Factor .....	893
Current Densities .....	894
Driving Forces .....	895
Total Anisotropic Mobility .....	896
Self-Consistent Anisotropic Mobility .....	896
Anisotropic Mobility Visualization .....	898
Anisotropic Metal Resistivity .....	898
Total Anisotropic Metal Resistivity Factor .....	899
Self-Consistent Anisotropic Metal Resistivity .....	899
Anisotropic Avalanche Generation .....	899
Anisotropic Electrical Permittivity .....	901
Anisotropic Thermal Conductivity .....	902
Anisotropic Density Gradient Model .....	903
Anisotropic Directions for Density Gradient Model .....	903
<b>29. Ferroelectric Materials .....</b>	<b>905</b>
Using Ferroelectrics .....	905

---

## Contents

---

Ferroelectrics Model . . . . .	908
Ginzburg–Landau–Khalatnikov Equation. . . . .	910
Using the Ginzburg–Landau–Khalatnikov Equation. . . . .	911
Versions of the Solver . . . . .	915
Default . . . . .	915
Multidimensional. . . . .	916
Solver II . . . . .	917
Bitlis Solver . . . . .	918
References. . . . .	920
<b>30. Ferromagnetism and Spin Transport</b> . . . . .	921
Brief Introduction to Spintronics . . . . .	921
Transport Through Magnetic Tunnel Junctions . . . . .	922
Magnetic Direct Tunneling Model. . . . .	922
Using the Magnetic Direct Tunneling Model. . . . .	923
Physics Parameters for Magnetic Direct Tunneling . . . . .	923
Math Parameters for Magnetic Direct Tunneling . . . . .	924
Magnetization Dynamics . . . . .	925
Spin Dynamics of a Free Electron in a Magnetic Field . . . . .	926
Magnetization Dynamics in a Ferromagnetic Layer. . . . .	926
Contributions of the Magnetic Energy Density . . . . .	927
Energy Density and Effective Field in Macrospin Approximation. . . . .	928
Using Magnetization Dynamics in Device Simulations . . . . .	929
Domain Selection and Initial Conditions. . . . .	929
Plotting Time-Dependent Magnetization . . . . .	930
Parameters for Magnetization Dynamics . . . . .	930
Time-Step Control for Magnetization Dynamics. . . . .	931
Thermal Fluctuations . . . . .	931
Using Thermal Fluctuations . . . . .	931
Parallel and Perpendicular Spin Transfer Torque . . . . .	932
Magnetization Dynamics Beyond Macrospin: Position-Dependent Exchange and Spin Waves . . . . .	932
Using Position-Dependent Exchange . . . . .	933
User-Defined Contributions to Effective Magnetic Field of LLG Equation . . . . .	934
References. . . . .	934

## Contents

---

<b>31. Mechanical Stress</b>	935
Overview of Mechanical Stress	935
Stress and Strain in Semiconductors	935
Using Stress and Strain	937
Stress Tensor	938
Strain Tensor	939
Stress Limits	939
Crystallographic Orientation and Compliance Coefficients	940
Deformation of Band Structure	940
Using Deformation Potential Model	943
Strained Effective Masses and Density-of-States	945
Strained Electron Effective Mass and DOS	945
Strained Hole Effective Mass and DOS	948
Using Strained Effective Masses and DOS	949
Multivalley Band Structure	950
Using Multivalley Band Structure	951
Mobility Modeling	952
Multivalley Electron Mobility Model	953
Intervalley Scattering	955
Effective Mass	956
Inversion Layer	958
Using Multivalley Electron Mobility Model	960
Multivalley Hole Mobility Model	963
Effective Mass	963
Scattering	964
Using Multivalley Hole Mobility Model	966
Multivalley Ballistic Mobility Model	968
Using Multivalley Ballistic Mobility Model	969
Multivalley Transferred Carrier Mobility Model	970
Using Multivalley Transferred Carrier Mobility Model	971
Intel Stress-Induced Hole Mobility Model	971
Stress Dependencies	973
Generalization of Model	973
Using Intel Mobility Model	975
Piezoresistance Mobility Model	976
Doping and Temperature Dependency	978
Using Piezoresistance Mobility Model	979
Named Parameter Sets for Piezoresistance	981
Auto-Orientation for Piezoresistance	981
Enormal- and MoleFraction-Dependent Piezo Coefficients	981
Using Piezoresistive Prefactors Model	982

## Contents

Isotropic Factor Models .....	988
Using Isotropic Factor Models .....	989
Piezoresistance Factor Models .....	989
Effective Stress Model .....	990
Mobility Stress Factor PMI Model.....	994
SFactor Dataset or PMI Model.....	994
Isotropic Factor Model Options .....	994
Factor Models Applied to Mobility Components.....	995
Stress Mobility Model for Minority Carriers .....	996
Dependency of Saturation Velocity on Stress .....	997
Mobility Enhancement Limits .....	998
Plotting Mobility Enhancement Factors .....	999
Numeric Approximations for Tensor Mobility .....	999
Tensor Grid Option.....	999
Stress Tensor Applied to Low-Field Mobility.....	1000
Piezoelectric Polarization.....	1001
Strain Model.....	1001
Simplified Strain Model.....	1002
Stress Model .....	1003
Poisson Equation.....	1003
Parameter File .....	1004
Coordinate Systems.....	1005
Converse Piezoelectric Field .....	1005
Piezoelectric Datasets .....	1006
Discontinuous Piezoelectric Charge at Heterointerfaces .....	1006
Gate-Dependent Polarization in GaN Devices.....	1006
Two-Dimensional Simulations .....	1007
Mechanics Solver.....	1008
Options for the Stress Solver.....	1009
References.....	1011
<hr/>	
32. Galvanic Transport .....	1015
Model Description .....	1015
Using Galvanic Transport Model .....	1015
Discretization Scheme for Continuity Equations .....	1016
References.....	1016

## Contents

---

<b>33. Thermal Properties</b>	1017
Heat Capacity	1017
The pmi_msc_heatcapacity Model	1018
Thermal Conductivity	1019
The AllDependent Thermal Conductivity Model	1020
Bulk Thermal Conductivity Computation	1021
Bulk Relaxation Time With Doping	1023
Thin-Layer Relaxation Time	1027
Mole Fraction–Dependent Relaxation Time	1028
The ConnallyThermalConductivity Model	1028
Layer Thickness Computation	1029
Bulk Thermal Conductivity Computation	1029
The pmi_msc_thermalconductivity Model	1031
Thermoelectric Power	1032
Physical Models	1032
Table-Based TEPower Model	1032
Analytic TEPower Model	1033
PMI_ThermoElectricPower Model	1034
Thermoelectric Power in Metals	1034
Using Thermoelectric Power	1034
Heating at Contacts, Metal–Semiconductor and Conductive Insulator–Semiconductor Interfaces	1035
References	1036
<b>34. Light-Emitting Diodes</b>	1038
Modeling Light-Emitting Diodes	1038
Coupling Electronics and Optics in LED Simulations	1039
Single-Grid Versus Dual-Grid LED Simulation	1039
Electrical Transport in Light-Emitting Diodes	1040
Spontaneous Emission Rate and Power	1040
Spontaneous Emission Power Spectrum	1041
Current File and Plot Variables for LED Simulations	1042
LED Wavelength	1045
Optical Absorption Heat	1045
Quantum Well Physics	1046
Accelerating Gain Calculations and LED Simulations	1047
Discussion of LED Physics	1047

## Contents

LED Optics: Raytracing . . . . .	1048
Compact Memory Raytracing . . . . .	1049
Isotropic Starting Rays From Spontaneous Emission Sources . . . . .	1049
Anisotropic Starting Rays From Spontaneous Emission Sources . . . . .	1050
Randomizing Starting Rays . . . . .	1051
Pseudorandom Starting Rays . . . . .	1052
Reading Starting Rays From File . . . . .	1052
Moving Starting Rays on Boundaries . . . . .	1053
Clustering Active Vertices . . . . .	1053
Plane Area Cluster . . . . .	1054
Nodal Clustering . . . . .	1055
Optical Grid Element Clustering . . . . .	1055
Using the Clustering Feature . . . . .	1055
Debugging Raytracing . . . . .	1055
Print Options in Raytracing . . . . .	1056
Interfacing LED Starting Rays to LightTools . . . . .	1057
Example: n99_000000_des_lighttools.txt File . . . . .	1059
LED Radiation Pattern . . . . .	1060
Two-Dimensional LED Radiation Pattern and Output Files . . . . .	1062
Three-Dimensional LED Radiation Pattern and Output Files . . . . .	1062
Staggered 3D Grid LED Radiation Pattern . . . . .	1063
Spectrum-Dependent LED Radiation Pattern . . . . .	1065
Tracing Source of Output Rays . . . . .	1066
Interfacing Far-Field Rays to LightTools . . . . .	1067
Example: farfield_lighttools.txt File . . . . .	1068
Nonactive Region Absorption (Photon Recycling) . . . . .	1068
Device Physics and Tuning Parameters . . . . .	1069
Example of 3D GaN LED Simulation . . . . .	1070
References . . . . .	1075
<hr/>	
35. Quantum Wells . . . . .	1076
Overview . . . . .	1076
Radiative Recombination and Gain Coefficients . . . . .	1077
Stimulated and Spontaneous Emission Coefficients . . . . .	1077
Active Bulk Material Gain . . . . .	1079
Stimulated Recombination Rate . . . . .	1079
Spontaneous Recombination Rate . . . . .	1080

## Contents

Fitting Stimulated and Spontaneous Emission Spectra.....	1080
Gain-Broadening Models.....	1080
Lorentzian Broadening.....	1080
Landsberg Broadening.....	1081
Hyperbolic-Cosine Broadening .....	1081
Syntax to Activate Broadening.....	1081
Electronic Band Structure for Wurtzite Crystals.....	1082
Optical Transition Matrix Element for Wurtzite Crystals.....	1086
Simple Quantum-Well Subband Model .....	1087
Syntax for Simple Quantum-Well Model .....	1090
Strain Effects .....	1090
Syntax for Quantum-Well Strain.....	1091
Localized Quantum-Well Model.....	1092
Nonlocal Quantum-Well Model Using 1D Schrödinger Solver.....	1095
Importing Gain and Spontaneous Emission Data With PMI .....	1096
Implementing Gain PMI .....	1097
References.....	1099
<hr/>	
<b>36. Kinetic Monte Carlo MIM Transport .....</b>	1101
KMC Simulation Space and Math Settings .....	1101
KMC Simulation Space .....	1102
Setting the Minimum Defect Distances .....	1103
Specifying the Random Seed.....	1103
Omitting KMC Simulations Near Steady State.....	1104
Omitting the KMC Simulation Completely .....	1104
Resetting the Fraction of Defects Filled With Electrons.....	1104
Sampling Potential at All Crossings .....	1105
Resetting the Rate More Frequently .....	1105
Adjusting the Calculated Tunneling Rates .....	1106
Setting the Integration Points.....	1106
Specifying the Approximate Bessel Function.....	1106
Specifying Electrodes .....	1107
Specifying Insulators .....	1107
Specifying Defects .....	1108
Defining Defects.....	1108
Reading in Defects.....	1110

## Contents

Creating the Grain Field . . . . .	1110
Tunneling Processes . . . . .	1112
Specifying Tunneling Processes and Related Options . . . . .	1112
Direct Tunneling . . . . .	1113
Poole–Frenkel Emission . . . . .	1114
Elastic Electrode-to-Trap and Trap-to-Electrode Tunneling . . . . .	1115
Option for Elastic Tunneling . . . . .	1116
Inelastic Electrode-to-Trap and Trap-to-Electrode Tunneling . . . . .	1116
Options for Inelastic Tunneling . . . . .	1118
Trap-to-Trap Tunneling . . . . .	1118
Trap-to-Trap Tunneling Model . . . . .	1118
Inelastic Phonon Trap-to-Trap Tunneling Model . . . . .	1118
Multiphonon Trap-to-Trap Tunneling Model . . . . .	1119
Image Charge Barrier Lowering . . . . .	1120
Energy-Dependent Mass . . . . .	1121
Including the MIM Charge in the Poisson Equation . . . . .	1122
Average Electron Occupancy in the MIM Charge . . . . .	1123
Printing Options . . . . .	1123
Rate Equation Current Calculation . . . . .	1123
Conductive Path Current Analysis . . . . .	1124
Sensitivity Analysis . . . . .	1125
Time-Dependent Dielectric Breakdown Simulations . . . . .	1128
Adjustment for Vacancy Sites per Volume . . . . .	1129
TDDB Break Criteria . . . . .	1130
TDDB Models . . . . .	1131
Fluence Dependency . . . . .	1132
Performing the Simulation . . . . .	1133
Printed Output . . . . .	1134
Setup Information . . . . .	1134
Occupancy and Energy . . . . .	1136
Event Statistics . . . . .	1137
Tunneling Rates . . . . .	1138
Quantities Available for Plotting . . . . .	1138
Quantities Written to the Current Plot (*.plt) File . . . . .	1138
Special MIM Current File (Currents and Capacitance) . . . . .	1139
Quantities Written to TDR Files . . . . .	1139
Concentration Plots . . . . .	1140
Particle Plots . . . . .	1140
Band Diagram With Traps . . . . .	1140

## Contents

---

Example: MIM Leakage Current .....	1141
Sentaurus Device Command File .....	1142
Sentaurus Device Parameter File .....	1144
Results .....	1144
Example: TiN/ZrO <sub>2</sub> /TiN KMC-TDDB Simulation .....	1145
Sentaurus Device Command File .....	1146
Sentaurus Device Parameter File .....	1148
Results .....	1149
References .....	1151
<b>37. Kinetic Monte Carlo ReRAM .....</b>	<b>1153</b>
Starting the Simulation and Specifying Particles .....	1153
Capacity of Defects .....	1154
KMC Events .....	1155
Location of KMC Events .....	1156
Particle or Particle Pairs Involved in KMC Events .....	1156
Parameters of KMC Events .....	1157
Diffusion Events .....	1157
Generation and Recombination Events .....	1157
Filament Growth and Recession Events .....	1158
KMC Simulation Space and Math Settings .....	1159
KMC Simulation Space .....	1159
Random Seed .....	1160
KMC Time Step Control .....	1160
Current Compliance .....	1161
Specifying the Material .....	1162
Specifying the Field and Temperature .....	1162
Transient Calculation for ReRAM .....	1163
Current in ReRAM Operations .....	1164
Conduction Current .....	1164
Trap-Assisted Tunneling Calculation .....	1164
Particles for Trap-Assisted Tunneling Calculation .....	1164
Physics for Trap-Assisted Tunneling Calculation .....	1165
Stopping Criteria of Trap-Assisted Tunneling Calculation .....	1165
Parameters for Trap-Assisted Tunneling Calculation .....	1166
Simulation Output .....	1167
Current and Particles .....	1167

## Contents

Event and Particle Statistics . . . . .	1167
Defect and Field Visualization . . . . .	1168
ReRAM Example . . . . .	1169
Sentaurus Device Command File . . . . .	1169
Sentaurus Device Parameter File . . . . .	1173

---

### Part III: Numeric Methods and External Interfaces

---

<b>38. Numeric Methods . . . . .</b>	1177
Discretization . . . . .	1177
Extended Precision . . . . .	1179
Box Method Coefficients in the 3D Case . . . . .	1179
Basic Definitions . . . . .	1179
Element Intersection Box Method Algorithm . . . . .	1182
Truncated Obtuse Elements . . . . .	1183
Weighted Box Method Coefficients . . . . .	1185
Weighted Points . . . . .	1185
Weighted Voronoï Diagram . . . . .	1186
Saving and Restoring Box Method Coefficients . . . . .	1187
Statistics About Non-Delaunay Elements . . . . .	1189
Region Non-Delaunay Elements . . . . .	1189
Interface Non-Delaunay Elements . . . . .	1190
Plot Section . . . . .	1190
Improved Accuracy of Box Method Parameters . . . . .	1191
AC Simulation . . . . .	1191
AC Response . . . . .	1192
AC Current Density Responses . . . . .	1194
Harmonic Balance Analysis . . . . .	1194
Harmonic Balance Equation . . . . .	1194
Multitone Harmonic Balance Analysis . . . . .	1195
Multidimensional Fourier Transformation . . . . .	1195
Quasi-Periodic Functions . . . . .	1196
Multidimensional Frequency Domain Problem . . . . .	1196
One-Tone Harmonic Balance Analysis . . . . .	1197
Solving the Harmonic Balance Equation . . . . .	1197
Solving the Harmonic Balance Newton Step Equation . . . . .	1198
Restarted GMRES Method . . . . .	1198
Direct Solver Method . . . . .	1199

## Contents

---

Transient Simulation . . . . .	1199
Backward Euler Method . . . . .	1199
TRBDF Composite Method . . . . .	1200
Controlling Transient Simulations . . . . .	1201
Floating Gates . . . . .	1202
Nonlinear Solvers . . . . .	1203
Fully Coupled Solution . . . . .	1203
‘Plugin’ Iterations . . . . .	1205
References . . . . .	1206
<b>39. Physical Model Interface . . . . .</b>	<b>1207</b>
Introduction to the Physical Model Interface . . . . .	1207
Using a PMI Model . . . . .	1210
Available Interfaces . . . . .	1210
Standard C++ Interface . . . . .	1211
Simplified C++ Interface . . . . .	1214
Nonlocal Interface . . . . .	1218
Shared Object Code . . . . .	1230
Command File of Sentaurus Device . . . . .	1230
Parameter File of Sentaurus Device . . . . .	1232
Parallelization . . . . .	1234
Thread-Local Storage . . . . .	1234
Debugging Physical Model Interfaces . . . . .	1235
Runtime Support for Vertex-Based PMI Models . . . . .	1237
Support at Model Scope . . . . .	1237
Support at Compute Scope . . . . .	1240
Experimental Functions . . . . .	1244
Runtime Support for Vertex-Based Multistate Configuration-Dependent Models . . . . .	1244
Runtime Support for Mesh-Based PMI Models . . . . .	1245
Device Mesh . . . . .	1247
Device Data . . . . .	1254
Runtime Support to Write PMI Output to Log File . . . . .	1259
Generation-Recombination . . . . .	1259
Avalanche Generation . . . . .	1259
Dependencies . . . . .	1260
Standard C++ Interface . . . . .	1261
Simplified C++ Interface . . . . .	1262
Example: Okuto Avalanche Generation . . . . .	1263

## Contents

Generation–Recombination . . . . .	1266
Dependencies . . . . .	1266
Standard C++ Interface . . . . .	1267
Simplified C++ Interface . . . . .	1268
Example: Auger Recombination . . . . .	1269
Lifetimes . . . . .	1269
Dependencies . . . . .	1269
Standard C++ Interface . . . . .	1270
Simplified C++ Interface . . . . .	1270
Example: Doping- and Temperature-Dependent Lifetimes . . . . .	1271
Nonlocal Generation–Recombination . . . . .	1273
Dependencies . . . . .	1273
Standard C++ Interface . . . . .	1274
Simplified C++ Interface . . . . .	1275
Example: Point-to-Point Tunneling . . . . .	1276
Tunneling Parameters . . . . .	1276
Dependencies . . . . .	1276
Simplified C++ Interface . . . . .	1277
Example: Constant Tunneling Parameters . . . . .	1277
Mobility . . . . .	1278
Doping-Dependent Mobility . . . . .	1279
Dependencies . . . . .	1279
Standard C++ Interface . . . . .	1280
Simplified C++ Interface . . . . .	1281
Example: Masetti Doping-Dependent Mobility . . . . .	1282
Fitting Parameter for Ballistic Mobility . . . . .	1284
Dependencies . . . . .	1285
Standard C++ Interface . . . . .	1285
Simplified C++ Interface . . . . .	1285
Example: Ballistic Mobility . . . . .	1286
High-Field Saturation . . . . .	1287
Dependencies . . . . .	1287
Standard C++ Interface . . . . .	1288
Simplified C++ Interface . . . . .	1290
Example: Canali High-Field Saturation . . . . .	1291
High-Field Saturation With Two Driving Forces . . . . .	1295
Command File . . . . .	1295
Dependencies . . . . .	1296
Standard C++ Interface . . . . .	1297
Simplified C++ Interface . . . . .	1299
Mobility Degradation at Interfaces . . . . .	1300
Dependencies . . . . .	1300
Standard C++ Interface . . . . .	1301

## Contents

Simplified C++ Interface . . . . .	1303
Example: Lombardi Mobility Degradation . . . . .	1304
Semiconductor Band Structure . . . . .	1308
Apparent Band-Edge Shift . . . . .	1308
Dependencies . . . . .	1309
Standard C++ Interface . . . . .	1310
Simplified C++ Interface . . . . .	1310
Band Gap . . . . .	1311
Dependencies . . . . .	1311
Standard C++ Interface . . . . .	1312
Simplified C++ Interface . . . . .	1312
Example: Default Temperature Dependence . . . . .	1313
Bandgap Narrowing . . . . .	1314
Dependencies . . . . .	1314
Standard C++ Interface . . . . .	1315
Simplified C++ Interface . . . . .	1315
Example: Bennett–Wilson Bandgap Narrowing . . . . .	1316
Effective Mass . . . . .	1317
Dependencies . . . . .	1317
Standard C++ Interface . . . . .	1317
Simplified C++ Interface . . . . .	1318
Example: Linear Effective Mass . . . . .	1319
Electron Affinity . . . . .	1320
Dependencies . . . . .	1321
Standard C++ Interface . . . . .	1321
Simplified C++ Interface . . . . .	1322
Example: Default Temperature Dependence . . . . .	1322
Phase and State Transitions . . . . .	1323
Multistate Configuration–Dependent Apparent Band-Edge Shift . . . . .	1323
Dependencies . . . . .	1324
Additional Functionality . . . . .	1325
Standard C++ Interface . . . . .	1325
Simplified C++ Interface . . . . .	1326
Multistate Configuration–Dependent Bulk Mobility . . . . .	1327
Command File . . . . .	1327
Dependencies . . . . .	1327
Standard C++ Interface . . . . .	1328
Simplified C++ Interface . . . . .	1329
Multistate Configuration–Dependent Heat Capacity . . . . .	1329
Command File . . . . .	1330
Dependencies . . . . .	1330
Standard C++ Interface . . . . .	1331
Simplified C++ Interface . . . . .	1331

## Contents

Multistate Configuration–Dependent Thermal Conductivity . . . . .	1332
Command File . . . . .	1332
Dependencies . . . . .	1333
Standard C++ Interface . . . . .	1334
Simplified C++ Interface . . . . .	1335
Thermal Properties and Heat . . . . .	1336
Distributed Thermal Resistance . . . . .	1336
Dependencies . . . . .	1336
Simplified C++ Interface . . . . .	1337
Example: Electron Density Dependence . . . . .	1338
Heat Capacity . . . . .	1339
Dependencies . . . . .	1339
Standard C++ Interface . . . . .	1340
Simplified C++ Interface . . . . .	1340
Example: Constant Heat Capacity . . . . .	1341
Heat Generation Rate . . . . .	1342
Dependencies . . . . .	1342
Standard C++ Interface . . . . .	1343
Simplified C++ Interface . . . . .	1344
Example: Dependency on Electric Field and Gradient of Temperature . . . . .	1345
Metal Thermoelectric Power . . . . .	1346
Dependencies . . . . .	1346
Standard C++ Interface . . . . .	1346
Simplified C++ Interface . . . . .	1347
Example: Linear Field Dependency of Metal TEP . . . . .	1348
Thermal Conductivity . . . . .	1350
Dependencies . . . . .	1350
Standard C++ Interface . . . . .	1351
Simplified C++ Interface . . . . .	1351
Example: Temperature-Dependent Thermal Conductivity . . . . .	1352
Example: Thin-Layer Thermal Conductivity . . . . .	1353
Thermoelectric Power . . . . .	1355
Dependencies . . . . .	1356
Standard C++ Interface . . . . .	1356
Simplified C++ Interface . . . . .	1357
Example: Analytic TEP . . . . .	1357
Optics . . . . .	1360
Complex Refractive Index Model Interface . . . . .	1360
C++ Application Programming Interface . . . . .	1361
Shared Object Code . . . . .	1366
Command File of Sentaurus Device . . . . .	1366
Optical Quantum Yield . . . . .	1367

## Contents

Dependencies .....	1367
Standard C++ Interface .....	1368
Simplified C++ Interface.....	1369
Special Contact PMI for Raytracing.....	1370
Dependencies .....	1371
Standard C++ Interface .....	1373
Example: Assessing and Modifying a Ray .....	1374
Mechanical Stress .....	1377
Mobility Stress Factor.....	1377
Dependencies .....	1377
Standard C++ Interface .....	1378
Simplified C++ Interface.....	1378
Example: Effective Stress .....	1379
Piezoelectric Polarization.....	1384
Dependencies .....	1384
Standard C++ Interface .....	1385
Simplified C++ Interface.....	1385
Example: Gaussian Polarization .....	1386
Piezoresistive Coefficients.....	1386
Dependencies .....	1387
Standard C++ Interface .....	1387
Simplified C++ Interface.....	1388
Stress.....	1388
Dependencies .....	1388
Standard C++ Interface .....	1389
Simplified C++ Interface.....	1390
Example: Constant Stress .....	1390
Traps and Fixed Charges .....	1392
Trap Capture and Emission Rates.....	1392
Traps .....	1392
Multistate Configurations .....	1392
Dependencies .....	1393
Standard C++ Interface .....	1394
Simplified C++ Interface.....	1395
Example: Arrhenius Law.....	1396
Trap Energy Shift.....	1396
Command File .....	1396
Dependencies .....	1396
Standard C++ Interface .....	1397
Simplified C++ Interface.....	1397
Degradation .....	1398
Diffusivity .....	1398

## Contents

Dependencies . . . . .	1399
Simplified C++ Interface . . . . .	1399
Example: Field-Dependent Hydrogen Diffusivity . . . . .	1400
eNMP Transition Rates . . . . .	1402
Distinction Between Electron and Hole Transitions . . . . .	1403
Transition Rates for All Sample Defects . . . . .	1403
Parameter Randomization . . . . .	1404
Dependencies . . . . .	1405
Standard C++ Interface . . . . .	1407
Simplified C++ Interface . . . . .	1408
Example: eNMP Model Transition Rates . . . . .	1409
Ferroelectrics and Ferromagnetics . . . . .	1413
Ferroelectrics . . . . .	1413
Ferroelectrics Hysteresis . . . . .	1415
Dependencies . . . . .	1415
Standard C++ Interface . . . . .	1416
Ferromagnetism and Spin Transport . . . . .	1417
User-Defined Interlayer Exchange Coupling . . . . .	1417
User-Defined Bulk or Interface Contributions to the Effective Magnetic Field . . . . .	1421
User-Defined Magnetostatic Potential Calculation . . . . .	1430
Electrical Resistivity . . . . .	1431
Metal Resistivity . . . . .	1431
Dependencies . . . . .	1431
Standard C++ Interface . . . . .	1432
Simplified C++ Interface . . . . .	1432
Example: Linear Metal Resistivity . . . . .	1433
Schottky Resistance . . . . .	1434
Dependencies . . . . .	1435
Standard C++ Interface . . . . .	1436
Simplified C++ Interface . . . . .	1437
Example: Built-in Schottky Resistance . . . . .	1437
Simulation Controls . . . . .	1439
Current Plot File . . . . .	1439
Structure of Current Plot File . . . . .	1440
Standard C++ Interface . . . . .	1440
Simplified C++ Interface . . . . .	1441
Example: Average Electrostatic Potential . . . . .	1442
Postprocessing for Transient Simulations . . . . .	1444
Standard C++ Interface . . . . .	1444
Simplified C++ Interface . . . . .	1444
Example: Postprocess User-Field . . . . .	1445

## Contents

Preprocessing for Newton Iterations and Newton Step Control . . . . .	1446
Standard C++ Interface . . . . .	1447
Simplified C++ Interface . . . . .	1448
Flow of Computation . . . . .	1449
Various . . . . .	1450
Dielectric Permittivity . . . . .	1450
Dependencies . . . . .	1450
Standard C++ Interface . . . . .	1451
Simplified C++ Interface . . . . .	1451
Example: Temperature-Dependent Dielectric Permittivity . . . . .	1452
Energy Relaxation Times . . . . .	1453
Dependencies . . . . .	1453
Standard C++ Interface . . . . .	1454
Simplified C++ Interface . . . . .	1454
Example: Constant Energy Relaxation Times . . . . .	1455
Gamma Factor for Density Gradient Model . . . . .	1457
Dependencies . . . . .	1457
Standard C++ Interface . . . . .	1458
Simplified C++ Interface . . . . .	1459
Example: Solution-Dependent Gamma Factor . . . . .	1460
Heavy Ion Spatial Distribution . . . . .	1462
Dependencies . . . . .	1462
Standard C++ Interface . . . . .	1462
Simplified C++ Interface . . . . .	1463
Example: Gaussian Spatial Distribution Function . . . . .	1464
Hot-Carrier Injection . . . . .	1465
Dependencies . . . . .	1465
Standard C++ Interface . . . . .	1465
Simplified C++ Interface . . . . .	1466
Example: Lucky Hot-Carrier Injection . . . . .	1467
Incomplete Ionization . . . . .	1472
Dependencies . . . . .	1472
Standard C++ Interface . . . . .	1473
Simplified C++ Interface . . . . .	1474
Example: Matsuura Incomplete Ionization . . . . .	1475
Space Factor . . . . .	1479
Dependencies . . . . .	1480
Standard C++ Interface . . . . .	1480
Simplified C++ Interface . . . . .	1481
Example: PMI User Field as Space Factor . . . . .	1481
References . . . . .	1482

## Contents

---

<b>40. Tcl Interfaces . . . . .</b>	1483
Overview . . . . .	1483
Mesh-Based Runtime Support. . . . .	1483
Device Mesh . . . . .	1484
Vertex . . . . .	1485
Edge . . . . .	1486
Element . . . . .	1487
Region . . . . .	1488
Region Interface. . . . .	1489
Device Data . . . . .	1489
One-Dimensional Arrays . . . . .	1491
Two-Dimensional Arrays. . . . .	1491
Current Plot File. . . . .	1491
Tcl Functions . . . . .	1491
tcl_cp_constructor . . . . .	1492
tcl_cp_destructor . . . . .	1492
tcl_cp_Compute_Dataset_Names . . . . .	1492
tcl_cp_Compute_Function_Names . . . . .	1493
tcl_cp_Compute_Plot_Values . . . . .	1493
Example: Average Electron Conductivity . . . . .	1493
<b>41. Python Interface. . . . .</b>	1496
Overview of Python Interface. . . . .	1496
Performing Device Simulations . . . . .	1497
Accessing Current Plot Data . . . . .	1497
Limitations . . . . .	1499

---

## Part IV: Appendices

---

<b>A. Mathematical Symbols . . . . .</b>	1501
<b>B. Syntax . . . . .</b>	1507
<b>C. File-Naming Conventions . . . . .</b>	1509
File Extensions. . . . .	1509

## Contents

---

<b>D. Command-Line Options</b> .....	1511
Starting Sentaurus Device .....	1511
Command-Line Options .....	1511
<b>E. Runtime Statistics</b> .....	1514
Generating Statistics .....	1514
<b>F. Data Names and Plot Names</b> .....	1515
Overview .....	1515
Scalar Data .....	1516
Vector Data .....	1552
Special Vector Data .....	1556
Tensor Data .....	1557
Particle Data .....	1557
<b>G. Command File Overview</b> .....	1558
Organization of Command File Overview .....	1558
Top Level of Command File .....	1560
CurrentPlot .....	1561
Device .....	1563
Electrode .....	1565
File .....	1567
GainPlot .....	1571
HydrogenBoundary .....	1572
IFM .....	1572
Math .....	1573
NoisePlot .....	1616
NonLocalPlot .....	1616
OpticalDevice .....	1616
Physics .....	1617
Generation and Recombination .....	1630

## Contents

LED .....	1655
Mobility.....	1662
Radiation Models .....	1668
Various .....	1669
Plot.....	1713
RayTraceBC .....	1714
Solve .....	1715
System.....	1735
TensorPlot .....	1737
Thermode.....	1738
Various.....	1738

# About This Guide

---

The Synopsys® Sentaurus™ Device tool is a full-featured, electrothermal, mixed-mode device and circuit simulator for one-dimensional, two-dimensional, and three-dimensional semiconductor devices. It incorporates advanced physical models and robust numeric methods for the simulation of most types of semiconductor device ranging from very deep-submicron silicon MOSFETs to large bipolar power structures. In addition, SiC and III-V compound homostructure and heterostructure devices are fully supported.

The user guide is divided into parts:

- Part I presents information about how to start Sentaurus Device and how it interacts with other Synopsys tools.
- Part II describes the physics and the physical models in Sentaurus Device.
- Part III describes the physical model interface, which provides direct access to certain models in the semiconductor transport equations, and the numeric methods used in Sentaurus Device.
- Part IV contains the appendices.

This guide assumes familiarity with working on UNIX-like systems and requires a basic understanding of semiconductor device physics and its terminology.

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNetPlus support site (see [Accessing SolvNetPlus on page 48](#))
- Documentation available on the SolvNetPlus support site

---

## Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
<b>Bold text</b>	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field or an option.
Courier font	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.

## About This Guide

Customer Support

Convention	Description
<i>Italicized text</i>	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
<b>Menu &gt; Command</b>	Indicates a menu command, for example, <b>File &gt; New</b> (from the <b>File</b> menu, choose <b>New</b> ).

---

## Customer Support

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

---

### Accessing SolvNetPlus

The SolvNetPlus support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNetPlus site:

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

---

### Contacting Synopsys Support

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

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

## About This Guide

### Acknowledgments

---

## Contacting Your Local TCAD Support Team Directly

Send an email message to:

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

## Acknowledgments

Parts of Sentaurus Device were codeveloped by Integrated Systems Laboratory of ETH Zurich in the joint research project LASER with financial support by the Swiss funding agency CTI and in the joint research project VCSEL with financial support by the Swiss funding agency TOP NANO 21. The GEBAS Library was codeveloped by Integrated Systems Laboratory of ETH Zurich in the joint research project MQW with financial support by the Swiss funding agency TOP NANO 21.

# **Part I:      Getting Started**

---

This part of the *Sentaurus™ Device User Guide* contains the following chapters:

- [Chapter 1, Introduction to Sentaurus Device](#)
- [Chapter 2, Specifying Physical Devices](#)
- [Chapter 3, Mixed-Mode Simulations](#)
- [Chapter 4, Performing Numeric Experiments](#)
- [Chapter 5, Simulation Results](#)
- [Chapter 6, Numeric and Software-Related Issues](#)

# 1

## Introduction to Sentaurus Device

---

*This chapter describes how Sentaurus Device integrates into the TCAD tool suite and presents some complete simulation examples.*

---

### Sentaurus Device Functionality

Sentaurus Device simulates numerically the electrical behavior of a single semiconductor device in isolation or several physical devices combined in a circuit. Terminal currents, voltages, and charges are computed based on a set of physical device equations that describes the carrier distribution and conduction mechanisms. A real semiconductor device, such as a transistor, is represented in the simulator as a ‘virtual’ device whose physical properties are discretized onto a nonuniform ‘grid’ (or ‘mesh’) of nodes.

Therefore, a virtual device is an approximation of a real device. Continuous properties such as doping profiles are represented on a sparse mesh and, therefore, are only defined at a finite number of discrete points in space. The doping at any point between nodes (or any physical quantity calculated by Sentaurus Device) can be obtained by interpolation. Each virtual device structure is described in the Synopsys TCAD tool suite by a TDR file containing the following information:

- The grid (or geometry) of the device contains a description of the various regions, that is, boundaries, material types, and the locations of any electrical contacts. It also contains the locations of all the discrete nodes and their connectivity.
- The data fields contain the properties of the device, such as the doping profiles, in the form of data associated with the discrete nodes. [Figure 1](#) shows a typical example: the doping profile of a MOSFET structure discretized by a mixed-element grid. By default, a device simulated in 2D is assumed to have a ‘thickness’ in the third dimension of 1  $\mu\text{m}$ .

The features of Sentaurus Device are many and varied. They can be summarized as:

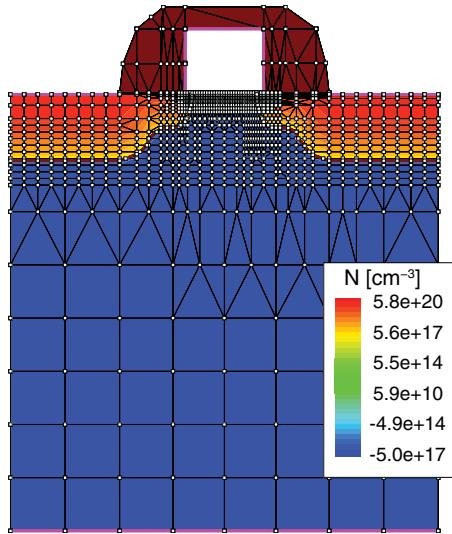
- An extensive set of models for device physics and effects in semiconductor devices (drift-diffusion, thermodynamic, and hydrodynamic models)
- General support for different device geometries (1D, 2D, 3D, and 2D cylindrical)

## Chapter 1: Introduction to Sentaurus Device

### Sentaurus Device Functionality

- Mixed-mode support of electrothermal netlists with mesh-based device models and SPICE circuit models

Figure 1 Two-dimensional doping profile that is discretized on the nodes of simulation grid

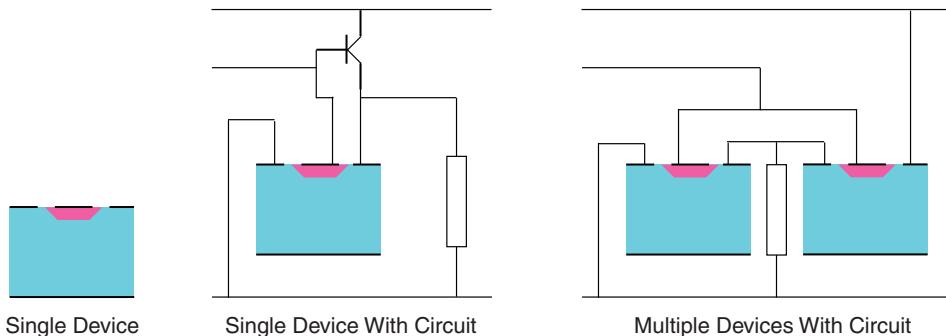


Nonvolatile memory simulations are accommodated by robust treatment of floating electrodes in combination with Fowler–Nordheim and direct tunneling, and hot-carrier injection mechanisms.

Hydrodynamic (energy balance) transport is simulated rigorously to provide a more physically accurate alternative to conventional drift-diffusion formulations of carrier conduction in advanced devices.

Floating semiconductor regions in devices such as thyristors and silicon-on-insulator (SOI) transistors (floating body) are handled robustly. This allows hydrodynamic breakdown simulations in such devices to be achieved with good convergence.

Figure 2 Three types of simulation



## Chapter 1: Introduction to Sentaurus Device

### Sentaurus Device Functionality

The mixed device and circuit capabilities give Sentaurus Device the ability to solve three basic types of simulation: single device, single device with a circuit netlist, and multiple devices with a circuit netlist (see [Figure 2](#)).

Multiple-device simulations can combine devices of different mesh dimensionality, and different physical models can be applied in individual devices, providing greater flexibility. In all cases, the circuit netlists can contain an electrical and a thermal section.

---

## Creating and Meshing Device Structures

Device structures can be created in various ways, including 1D, 2D, or 3D process simulation (Sentaurus Process), 2D or 3D process emulation (Sentaurus Structure Editor), and 2D or 3D structure editors (Sentaurus Structure Editor).

Regardless of the means used to generate a virtual device structure, it is recommended that the structure be remeshed using Sentaurus Structure Editor (2D and 3D meshing with an interactive graphical user interface (GUI)) or Sentaurus Mesh (1D, 2D, and 3D meshing without a GUI) to optimize the grid for efficiency and robustness.

For maximum efficiency of a simulation, a mesh must be created with a minimum number of vertices to achieve the required level of accuracy. For any given device structure, the optimal mesh varies depending on the type of simulation.

It is recommended that to create the most suitable mesh, the mesh must be densest in those regions of the device where the following are expected:

- High current density (MOSFET channels, bipolar base regions)
- High electric fields (MOSFET channels, MOSFET drains, depletion regions in general)
- High charge generation (single event upset (SEU) alpha particle, optical beam)

For example, accurate drain current modeling in a MOSFET requires very fine, vertical, mesh spacing in the channel at the oxide interface (of the order 1 Å) when using advanced mobility models. For reliable simulation of breakdown at a drain junction, the mesh must be more concentrated inside the junction depletion region for good resolution of avalanche multiplication.

Generally, a total node count of 2000 to 4000 is reasonable for most 2D simulations. Large power devices and 3D structures require a considerably larger number of elements.

---

## Tool Flow

In a typical device tool flow, the creation of a device structure by process simulation (Sentaurus Process) is followed by remeshing using Sentaurus Structure Editor or

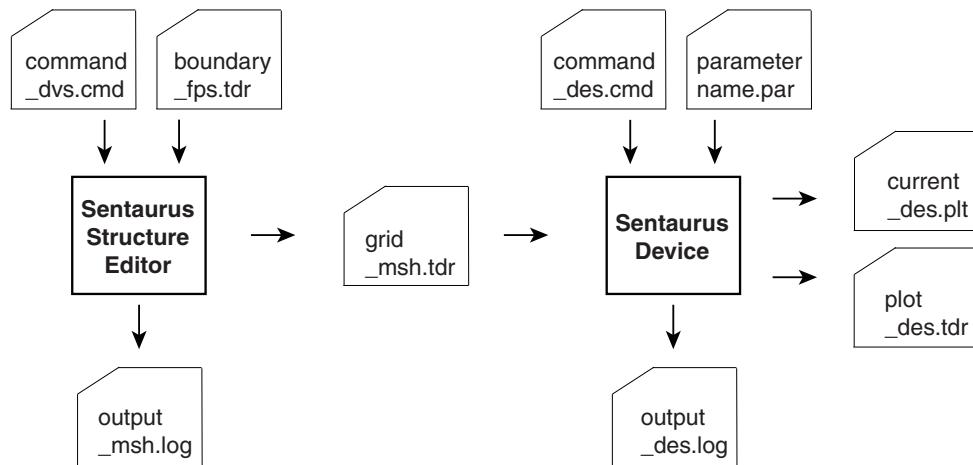
## Chapter 1: Introduction to Sentaurus Device

### Sentaurus Device Functionality

Sentaurus Mesh. In this scheme, control of mesh refinement is handled automatically through the file `_dvs.cmd`.

Sentaurus Device is used to simulate the electrical characteristics of the device. Finally, Sentaurus Visual is used to visualize the output from the simulation in 2D and 3D, and Inspect is used to plot the electrical characteristics.

*Figure 3 Typical tool flow with device simulation using Sentaurus Device*



---

## Generating Input Files for Sentaurus Device

You can generate input command files for Sentaurus Device by using the Sentaurus Device Wizard, which is part of the Sentaurus Workbench software package.

The Sentaurus Device Wizard provides a graphical user interface that guides you through various steps to generate Sentaurus Device inputs to simulate various applications, such as silicon-based diodes, transistors, and memory, as well as other device types (such as SiGe transistors, III-V group compounds, wide-bandgap devices, ferroelectrics, organics, and passive photonics).

See *Sentaurus™ Workbench User Guide*, Generating Input Files for Sentaurus Device.

For more information about the Sentaurus Device Wizard, go to the TCAD Sentaurus Tutorial, Sentaurus Workbench module, Section 3.3, Using the Sentaurus Device Wizard (see [TCAD Sentaurus Tutorial: Simulation Projects on page 56](#)).

## Chapter 1: Introduction to Sentaurus Device

### Starting Sentaurus Device

---

## Starting Sentaurus Device

You can start Sentaurus Device from either the command line or Sentaurus Workbench.

---

### From the Command Line

Sentaurus Device is driven by a command file and run by the command:

```
sdevice <command_filename>
```

Various options exist at start-up and are listed by using:

```
sdevice -h
```

By default, `sdevice` runs the latest version in the current release of Sentaurus Device. To run a particular version in the current release, use the `-ver` command-line option. The following example starts version 1.4 of the latest release of Sentaurus Device:

```
sdevice -ver 1.4 nmos_des.cmd
```

To run the latest version in a particular release, use the `-rel` command-line option. The following example starts the latest version available in release T-2022.03:

```
sdevice -rel T-2022.03 nmos_des.cmd
```

To run a particular version in a particular release, combine `-ver` and `-rel`. The following example starts Sentaurus Device, release S-2021.06, version 1.2:

```
sdevice -rel S-2021.06 -ver 1.2 nmos_des.cmd
```

When the release or the version requested is not installed, the available releases or versions are listed, and the program terminates.

[Appendix D on page 1511](#) lists the command options of Sentaurus Device, which include:

```
sdevice -versions
```

Checks which versions are in the installation path.

```
sdevice -P and sdevice -L
```

Extracts model parameter files (see [Generating a Copy of Parameter File on page 80](#)).

```
sdevice --parameter-names
```

Prints names of parameters that can be ramped (see [Ramping Physical Parameter Values on page 126](#)).

When Sentaurus Device starts, the command file is checked for correct syntax, and the commands are executed in sequence. Character strings starting with \* or # are ignored by

## Chapter 1: Introduction to Sentaurus Device

TCAD Sentaurus Tutorial: Simulation Projects

Sentaurus Device, so that these characters can be used to insert comments in the simulation command file.

---

## From Sentaurus Workbench

Sentaurus Device is launched automatically through the Scheduler when working inside Sentaurus Workbench.

Sentaurus Workbench interprets # as a special marker for conditional statements. For example:

```
#if...  
#elif...  
#endif...
```

---

## TCAD Sentaurus Tutorial: Simulation Projects

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

To access the TCAD Sentaurus Tutorial:

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

Alternatively, to access the TCAD Sentaurus Tutorial:

1. Go to the \$STROOT/tcad/current/Sentaurus\_Training directory.

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

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

# 2

## Specifying Physical Devices

---

*This chapter describes how to specify physical devices.*

Before performing a simulation, Sentaurus Device needs to know which device will actually be simulated. This chapter describes what Sentaurus Device needs to know and how to specify it. This includes obvious information, such as the size and shape of the device, the materials of which the device is made, and the doping profiles. It also includes less obvious aspects such as which physical effects must be taken into account, and by which models and model parameters this should be performed.

---

### Reading a Structure

A device is defined by its shape, material composition, and doping. This information is defined on a grid and contained in a TDR file that you specify with the keyword `Grid` in the `File` section of the command file. For example:

```
File {
    Grid = "mosfet.tdr"
    ...
}
```

The following keywords affect the interpretation of the geometry in the `Grid` file:

- `CoordinateSystem` in the global `Math` section specifies the coordinate system that is used for explicit coordinates in the Sentaurus Device command file. Many features such as current plot statements (see [Tracking Additional Data in the Current File on page 165](#)) or `LatticeParameters` in the parameter file (see [Crystal and Simulation Coordinate Systems on page 886](#)) use explicit coordinates, and they are based on an implicit assumption regarding the orientation of the device. You can indicate the required orientation of the device by specifying the following keyword:

```
Math {
    CoordinateSystem { <option> }
}
```

## Chapter 2: Specifying Physical Devices

### Reading a Structure

If the coordinate system of the device is incompatible with the specification in the `Math` section, then Sentaurus Device automatically transforms the device into the required coordinate system.

*Table 1 Options of the CoordinateSystem keyword*

Option	Description
AsIs	(Default) This option specifies that the coordinate system of the device is compatible with the explicit coordinates used in the Sentaurus Device command file. No transformation is applied to the structure.
DFISE	In two dimensions, the x-axis points across the surface, and the y-axis points down into the device. In 3D, the x-axis and y-axis span the surface of the device, and the z-axis points upwards away from the device.
UCS	The unified coordinate system (UCS) uses the convention of the simulation coordinate system as established by Sentaurus Process. The x-axis always points down into the device. In 2D, the y-axis runs across the surface of the device. In 3D, the z-axis is added to obtain a right-handed coordinate system.

- `AreaFactor` in the global `Physics` section specifies a multiplier for currents and charges. For 1D or 2D simulations, it typically specifies the extension of the device in the remaining one or two dimensions. In simulations that exploit the symmetry of the device, it can also be used to account for the reduction of the simulated device compared to the real device. `AreaFactor` is also available in the `Electrode` and `Thermode` sections, with the same meaning; if both `AreaFactors` are present, Sentaurus Device multiplies them.

TDR units are taken into account when loading from a `.tdr` file. Thereby, the units read from files are converted to the appropriate units used in Sentaurus Device. The TDR unit is ignored in the case of a conversion failure. Use the keyword `IgnoreTdrUnits` in the `Math` section to disregard TDR units during loading. This applies not only to the `Grid` file, but also to other loaded files (see [Save and Load Statements on page 213](#)).

---

## Cylindrical Coordinate System

The cylindrical coordinate system models the radial ( $\rho$ ) and height ( $z$ ) dependency of 3D devices with cylindrical symmetry, or models devices or problems with circular cross-sections. The device behavior is assumed to be identical for all values of the azimuthal angle ( $\phi$ ). Therefore, a 3D device can be described in a 2D coordinate system.

Sentaurus Device assumes that the 2D structure is rotated completely about the cylindrical  $z$ -axis through an angle of  $2\pi$  radians. As a result, a 2D mesh can be used to simulate a 3D cylindrical device, and the simulation results are equivalent to a 3D simulation. Therefore,

## Chapter 2: Specifying Physical Devices

### Reading a Structure

the unit of the terminal current is ampere (A) *not A/ $\mu\text{m}$* , which is the unit of the terminal current for 2D device structures.

You activate cylindrical coordinates by specifying the keyword `Cylindrical` in the global `Math` section (only applies to 2D meshes):

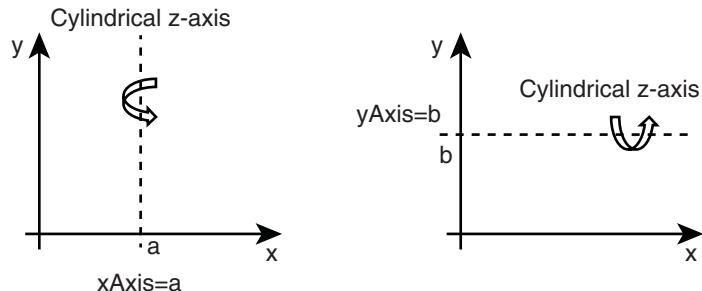
```
Math {  
    Cylindrical( xAxis=<float> )      # or yAxis=<float>  
}
```

You can define where the cylindrical z-axis is located by using the option `xAxis=<float>` or `yAxis=<float>`. [Figure 4](#) explains the usage of these options and pertains to DF-ISE coordinates. In the case of the UCS, the x-axis is the vertical axis, and the y-axis is the horizontal axis. Therefore, you need to set the options `xAxis` and `yAxis` carefully to define their cylindrical z-axis.

**Note:**

No contacts should be aligned to the cylindrical z-axis. Otherwise, an error will terminate the simulation.

*Figure 4 Cylindrical z-axis can be shifted along the axis of definition with the `xAxis` or `yAxis` option*



---

## Abrupt and Graded Heterojunctions

Sentaurus Device supports both abrupt and graded heterojunctions, with an arbitrary mole fraction distribution. In the case of abrupt heterojunctions, Sentaurus Device treats discontinuous datasets properly by introducing double points at the heterointerfaces.

This option is switched on automatically when thermionic emission (see [Thermionic Emission Current on page 870](#)) is selected, or when the keyword `HeteroInterface` is specified in the `Physics` section of a selected heterointerface. By default, this double points option is switched off.

## Chapter 2: Specifying Physical Devices

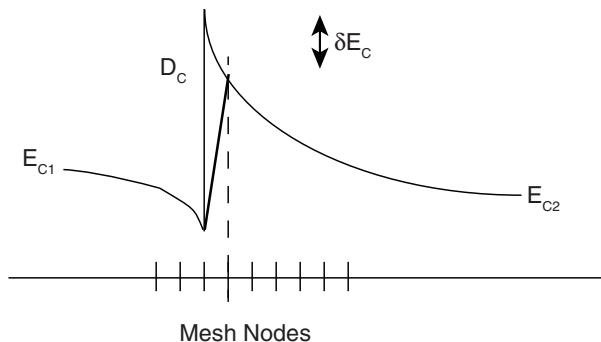
### Specifying Doping Species

#### Note:

The keyword `HeteroInterface` provides equilibrium conditions (continuous quasi-Fermi potentials) for the double points. It does not provide realistic physics at the interface for high-current regimes. Using the `HeteroInterface` option without thermionic emission or a tunneling model is discouraged.

To illustrate the double points option, [Figure 5](#) shows the conduction band near an abrupt heterointerface. The wide line shows a case without double points, which requires a very fine mesh to avoid a large barrier error ( $\delta E_C$ ).

*Figure 5 Band edge at a heterointerface with and without double points*



---

## Specifying Doping Species

Sentaurus Device relies on the variables section of the file `datexcodes.txt` to determine its doping species. A variable is identified as a doping species by a doping field that shows whether it is an acceptor or a donor. Chemical concentrations are linked to their corresponding active concentrations by an `active` field. Similarly, ionized concentrations are specified by an `ionized` field. A typical declaration would be:

```
BoronConcentration, BoronChemicalConcentration {
    doping = acceptor (
        active = BoronActiveConcentration
        ionized = BoronMinusConcentration
    )
    ...
}

BoronActiveConcentration { ... }

BoronMinusConcentration { ... }
```

## Chapter 2: Specifying Physical Devices

### Specifying Doping Species

You can also limit the doping species to certain substrate materials. For example, to restrict `SiliconConcentration` as a donor for GaN substrates only, specify the following:

```
SiliconConcentration, SiliconChemicalConcentration {  
    doping = donor (  
        active = SiliconActiveConcentration  
        ionized = SiliconPlusConcentration  
        material = GaN  
    )  
}
```

The TCAD Sentaurus tool suite provides a default `datexcodes.txt` file in the directory `$STROOT_LIB` (or `$STROOT/tcad/$STRELEASE/lib`). Many common doping species are already predefined in this file. To add user-defined doping species or to modify an existing specification, you can use a `datexcodes.txt` file in the local directory. The local `datexcodes.txt` file only needs to contain the variables that you want to add or modify.

If the incomplete ionization model is activated (see [Using the Incomplete Ionization Model on page 339](#)), the model parameters of the user-defined species must be specified in the `Ionization` section of the material parameter file.

Sentaurus Device loads doping distributions from the `Grid` file specified in the `File` section (see [Reading a Structure on page 57](#)) and reads the following datasets:

- Net doping (`DopingConcentration` in the `Grid` file)
- Total doping (`TotalConcentration` in the `Grid` file)
- Concentrations of individual species

Sentaurus Device supports the following rules of doping specification:

- Sentaurus Device takes the net doping dataset from the file if this dataset is present. Otherwise, net doping is recomputed from the concentrations of the separate species. To force the recomputation of the net doping based on individual species, use the keyword `ComputeDopingConcentration` in the global `Math` section.
- The same rule applies to the total doping dataset.

**Note:**

Total concentration, which originates from process simulators such as Sentaurus Process, is the sum of the chemical concentrations of dopants. However, if the total concentration is recomputed inside Sentaurus Device, active concentrations are used.

- Sentaurus Device takes the active concentration of a dopant if it is in the `Grid` file (for example, `BoronActiveConcentration`). Otherwise, the chemical dopant concentration is used (for example, `BoronConcentration`).

## Chapter 2: Specifying Physical Devices

### Specifying Materials

To perform any simulation, Sentaurus Device must prepare the following major doping arrays:

- Net doping concentration,  $N_{\text{net}} = N_{D,0} - N_{A,0}$
- Donor and acceptor concentrations,  $N_{D,0}$  and  $N_{A,0}$
- Total doping concentration,  $N_{\text{tot}} = N_{D,0} + N_{A,0}$

After loading the doping file, Sentaurus Device uses the following scheme to compute the doping arrays:

1. If the `Grid` file does not have any species,  $N_{\text{net}}$  is initialized from `DopingConcentration`,  $N_{\text{tot}}$  is initialized from `TotalConcentration` or  $|N_{\text{net}}|$  if `TotalConcentration` was not read,  $N_{D,0} = (N_{\text{tot}} + N_{\text{net}})/2$ , and  $N_{A,0} = (N_{\text{tot}} - N_{\text{net}})/2$ .
2. If the `Grid` file has individual species,  $N_{A,0}$  and  $N_{D,0}$  are computed as the sum of individual donor and acceptor concentrations,  $N_{\text{net}}$  is initialized from `DopingConcentration` or  $N_{D,0} - N_{A,0}$  if `DopingConcentration` was not read, and  $N_{\text{tot}}$  is initialized from `TotalConcentration` or  $N_{A,0} + N_{D,0}$  if `TotalConcentration` was not read.
3. Doping concentration is also affected if the `Physics` section includes trap specifications with the `Add2TotalDoping` or `Add2TotalDoping(ChargedTraps)` option (see [Options to Include Traps in Doping on page 552](#) for a description of these options).

---

## Specifying Materials

Sentaurus Device supports all materials that are declared in the `datexcodes.txt` file (see *Utilities User Guide*, Chapter 1). The following search strategy is observed to locate the `datexcodes.txt` file:

- `$STROOT_LIB/datexcodes.txt` or `$STROOT/tcad/$STRELEASE/lib/datexcodes.txt` if the environment variable `STROOT_LIB` is not defined (lowest priority)
- `$HOME/datexcodes.txt` (medium priority)
- `datexcodes.txt` in local directory (highest priority)

Definitions in later files replace or add to the definitions in the earlier files. In this way, the local file only needs to contain materials or variables that you want to add or modify.

---

## User-Defined Materials

New materials can be defined in a local `datexcodes.txt` file. To add a new material, add its description to the `Materials` section of `datexcodes.txt`:

```
Materials {
    Silicon {
        label = "Silicon"
        group = Semiconductor
        color = #ffb6c1
    }
    Oxide {
        label = "SiO2"
        group = Insulator
        color = #7d0505
    }
    ...
}
```

The `label` value is used as a legend in visualization tools.

The `group` value identifies the type of the new material. The available options are `All`, `Conductor`, `Insulator`, or `Semiconductor`.

The field `color` defines the color of the material in visualization tools and must have the following syntax:

```
color = #rrggbb
```

Here, `rr`, `gg`, and `bb` denote hexadecimal numbers representing the intensity of red, green, and blue, respectively. The values of `rr`, `gg`, and `bb` must be in the range `00` to `ff`.

*Table 2      Sample values for color*

Color code	Color	Color code	Color
#000000	Black	#ffffff	White
#ff0000	Red	#40e0d0	Turquoise
#00ff00	Green	#7fff00	Chartreuse
#0000ff	Blue	#b03060	Maroon
#ffff00	Yellow	#ff7f50	Coral
#ff00ff	Magenta	#da70d6	Orchid
#00ffff	Cyan	#e6e6fa	Lavender

## Chapter 2: Specifying Physical Devices

### Specifying Materials

---

## Mole-Fraction Materials

Sentaurus Device reads the `Molefraction.txt` file to determine mole fraction-dependent materials. The following search strategy is used to locate this file:

1. Sentaurus Device looks for `Molefraction.txt` in the current working directory.
2. If the environment variables `STROOT` and `STRELEASE` are defined, Sentaurus Device tries to read the `$STROOT/tcad/$STRELEASE/lib/Molefraction.txt` file.
3. If these previous strategies are unsuccessful, Sentaurus Device uses the built-in defaults that follow.

The following example shows the default `Molefraction.txt` file (but you are urged to check `$STROOT/tcad/$STRELEASE/lib/Molefraction.txt` for the latest version of this file):

```
# Ge(x)Si(1-x)
SiliconGermanium (x=0) = Silicon
SiliconGermanium (x=1) = Germanium

# Al(x)Ga(1-x)As
AlGaAs (x=0) = GaAs
AlGaAs (x=1) = AlAs

# In(1-x)Al(x)As
InAlAs (x=0) = InAs
InAlAs (x=1) = AlAs

# In(1-x)Ga(x)As
InGaAs (x=0) = InAs
InGaAs (x=1) = GaAs

# Ga(x)In(1-x)P
GaInP (x=0) = InP
GaInP (x=1) = GaP

# InAs(x)P(1-x)
InAsP (x=0) = InP
InAsP (x=1) = InAs

# GaAs(x)P(1-x)
GaAsP (x=0) = GaP
GaAsP (x=1) = GaAs

# Hg(1-x)Cd(x)Te
HgCdTe (x=0) = HgTe
HgCdTe (x=1) = CdTe

# In(1-x)Ga(x)As(y)P(1-y)
InGaAsP (x=0, y=0) = InP
InGaAsP (x=1, y=0) = GaP
```

## Chapter 2: Specifying Physical Devices

### Mole-Fraction Specification

```
InGaAsP (x=1, y=1) = GaAs  
InGaAsP (x=0, y=1) = InAs
```

To add a new mole fraction-dependent material, the material (and its side and corner materials) must first be added to `datexcodes.txt`. Afterwards, `Molefraction.txt` can be updated.

Quaternary alloys are specified by their corner materials in the file `Molefraction.txt`. For example, the 2:2 III-V quaternary alloy  $\text{In}_{1-x}\text{Ga}_x\text{As}_y\text{P}_{1-y}$  is given by:

```
InGaAsP (x=0, y=0) = InP  
InGaAsP (x=1, y=0) = GaP  
InGaAsP (x=1, y=1) = GaAs  
InGaAsP (x=0, y=1) = InAs
```

The 3:1 III-V quaternary alloy  $\text{Al}_x\text{In}_y\text{Ga}_{1-x-y}\text{As}$  is defined by:

```
AlInGaAs (x=0, y=0) = GaAs  
AlInGaAs (x=1, y=0) = AlAs  
AlInGaAs (x=0, y=1) = InAs
```

When the corner materials of an alloy have been specified, Sentaurus Device determines the corresponding side materials automatically. In the case of  $\text{In}_{1-x}\text{Ga}_x\text{As}_y\text{P}_{1-y}$ , the four side materials are  $\text{InAs}_x\text{P}_{1-x}$ ,  $\text{GaAs}_x\text{P}_{1-x}$ ,  $\text{Ga}_x\text{In}_{1-x}\text{P}$ , and  $\text{In}_{1-x}\text{Ga}_x\text{As}$ .

Similarly, for  $\text{Al}_x\text{In}_y\text{Ga}_{1-x-y}\text{As}$ , the three side materials are  $\text{In}_{1-x}\text{Al}_x\text{As}$ ,  $\text{Al}_x\text{Ga}_{1-x}\text{As}$ , and  $\text{In}_{1-x}\text{Ga}_x\text{As}$ .

#### Note:

All side and corner materials must appear in the `datexcodes.txt` file, and their mole dependencies must be specified in `Molefraction.txt` (see [Specifying Materials on page 62](#)).

If it cannot parse the `Molefraction.txt` file, Sentaurus Device reverts to the defaults shown previously. This might lead to unexpected simulation results.

---

## Mole-Fraction Specification

In Sentaurus Device, the mole fraction of a compound semiconductor or insulator is defined in the following ways:

- In the `Grid` file (`<name>.tdr`) of the device structure
- Internally, in the `Physics` section of the command file

If the mole fraction is loaded from the `.tdr` file and an internal mole fraction specification is also applied, the loaded mole fraction values are overwritten in the regions specified in the `MoleFraction` sections of the command file.

## Chapter 2: Specifying Physical Devices

### Physical Models and the Hierarchy of Their Specification

The internal mole fraction distribution is described in the `MoleFraction` statement inside the `Physics` section:

```
Physics { ...
    MoleFraction(<MoleFraction parameters>)
}
```

The parameters for the mole fraction specification and grading options are described in [Table 316 on page 1691](#).

The specification of an `xFraction` is mandatory in the `MoleFraction` statement for binary or ternary compounds; a `yFraction` is also mandatory for quaternary materials. If the `MoleFraction` statement is inside a default `Physics` section, the `RegionName` must be specified. If it is inside a region-specific `Physics` section, by default, it is applied only to that region. If a `MoleFraction` statement is inside a material-specific `Physics` section and the `RegionName` is not specified, this composition is applied to all regions containing the specified material. If `RegionName` is specified inside a region-specific and material-specific `Physics` section, this specification is used instead of the default regions.

#### Note:

Similar to all statements, only one `MoleFraction` statement is allowed inside each `Physics` section. By default, grading is not included.

An example of a mole fraction specification is:

```
Physics {
    MoleFraction(RegionName = [ "Region.3" "Region.4" ]
        xFraction=0.8
        yFraction=0.7
        Grading(
            (xFraction=0.3 GrDistance=1
                RegionInterface= ("Region.0" "Region.3" ))
            (xFraction=0.2 yFraction=0.1 GrDistance=1
                RegionInterface= ("Region.0" "Region.5" ))
            (yFraction=0.4 GrDistance=1
                RegionInterface= ("Region.0" "Region.3" ))
        )
    )
}
Physics (Region = "Region.6") {
    MoleFraction(xFraction=0.1 yFraction=0.7 GrDistance=0.01)
}
```

---

## Physical Models and the Hierarchy of Their Specification

The `Physics` section is used to select the models that are used to simulate a device. [Table 236 on page 1617](#) lists the keywords that are available, and Part II discusses the

## Chapter 2: Specifying Physical Devices

### Physical Models and the Hierarchy of Their Specification

models in detail. Physical models can be specified globally, per region or material, per interface, or per electrode.

Some models (for example, the hydrodynamic transport model) can only be activated for the whole device. Regionwise or materialwise specifications are syntactically possible, but Sentaurus Device silently ignores them. Likewise, some specifications are syntactically possible for all locations, but they are semantically valid only for interfaces or bulk regions. In the tables in [Appendix G on page 1558](#), the validity of specifications is indicated in the description column by characters in parentheses. For example, (g) denotes models that can be activated only for the entire device, but not for individual parts of it.

Some specifications are syntactically possible everywhere, but are valid for certain materials only. For example, `Mobility` is only valid in semiconductors.

---

## Region-Specific and Material-Specific Models

In Sentaurus Device, different physical models for different regions and materials within a device structure can be specified. The syntax for this feature is:

```
Physics (material="material") {  
    <physics-body>  
}
```

or:

```
Physics (region="region-name") {  
    <physics-body>  
}
```

This feature is also available for the `Math` section:

```
Math (material="material") {  
    <math-body>  
}
```

or:

```
Math (region="region-name") {  
    <math-body>  
}
```

A `Physics` section without any region or material specifications is considered the default section.

#### Note:

You can edit region names in Sentaurus Structure Editor (see *Sentaurus™ Structure Editor User Guide*, Changing the Name of a Region).

## Chapter 2: Specifying Physical Devices

### Physical Models and the Hierarchy of Their Specification

The `Physics` and `Math` sections for different locations are related as follows:

- Physical models defined in the global `Physics` section (that is, in the section without any region or material specifications) are applied in all regions of the device.
- Physical models defined in a material-specific `Physics` section are added to the default models for all regions containing the specified material.
- The same applies to the physical models defined in a region-specific `Physics` section: all regionwise defined models are added to the models defined in the default section.

#### Note:

If for a region, both a region-specific `Physics` section and a material-specific `Physics` section for the material of the region are present, then the region-specific declaration overrides the material-specific declaration, that is, the region-specific `Physics` section will not inherit any models from the material-specific section.

For example:

```
Physics {<Default models>}
Physics (Material="GaAs") {<GaAs models>}
Physics (Region="Emitter") {<Emitter models>}
```

If the "Emitter" region is made of GaAs, the models in this region are `<Default models>` and `<Emitter models>`, that is, whatever `<GaAs models>` contains is ignored in region "Emitter".

For some models, the model specification and numeric values of the parameters are defined in the `Physics` sections. Examples of such models are `Traps` and the `MoleFraction` specifications.

For these models, the specifications in region or material `Physics` sections overwrite previously defined values of the corresponding parameters.

If in the default `Physics` section, `MoleFraction` is defined for a given region and, afterward, is defined for the same region again, in a region or material `Physics` section, then the default definition is overwritten. The hierarchy of the parameter specification is the same as discussed previously.

---

## Interface-Specific Models

A special set of models can be activated at the interface between two different materials or two different regions. In [Table 236 on page 1617](#), pure interface models are flagged with '(i)' in the description column.

## Chapter 2: Specifying Physical Devices

### Physical Models and the Hierarchy of Their Specification

As physical phenomena at an interface are not the same as in the bulk of a device, not all models are allowed inside interface-specific `Physics` sections. For example, it is not possible to define any mobility models or bandgap narrowing at interfaces.

#### Note:

Although the `Recombination(surfaceSRH)` statement and the `GateCurrent` statement describe pure interface phenomena, they can be defined in a region-specific `Physics` section. In this case, the models are applied to all interfaces between this region and all adjacent insulator regions. If specified in the global `Physics` section, these models are applied to all semiconductor-insulator interfaces.

Interface models are specified in interface-specific `Physics` sections. Their respective parameters are accessible in the parameter file. The syntax for specification of an interface model is:

```
Physics (MaterialInterface="material-name1/material-name2") {  
    <physics-body>  
}
```

or:

```
Physics (RegionInterface="region-name1/region-name2") {  
    <physics-body>  
}
```

The following is an example illustrating the specification of fixed charges at the interface between the materials oxide and aluminum gallium arsenide (AlGaAs):

```
Physics(MaterialInterface="Oxide/AlGaAs") {  
    Traps(Conc=-1.e12 FixedCharge)  
}
```

If no region interface `Physics` section is present for a given region interface, the material interface section is used if present. If the material interface `Physics` section is missing as well, built-in defaults are used. Similar to what holds for regions and materials, if a region interface section is present, it is used for the region interface, ignoring material interface settings completely, even when a material interface `Physics` section is present. However, other than for regions and materials, the global `Physics` section is not used automatically to determine interface `Physics` settings.

---

## Electrode-Specific Models

Electrode-specific `Physics` sections can be defined. For example:

```
Physics(Electrode="Gate") {  
    Schottky  
    eRecVel = <float>
```

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

```
    hRecVel = <float>
    Workfunction = <float>
}
```

---

## Physical Model Parameters

Most physical models depend on parameters that can be adjusted in a file given by `Parameter` in the `File` section:

```
File {
    Parameter = <string>
    ...
}
```

The name of the parameter file conventionally has the extension `.par`.

Model parameters are split into sets, where a particular set corresponds to a particular physical model. The available parameter sets and the individual parameters they contain are described along with the description of the related model (see Part II).

Parameters can be specified globally, materialwise, regionwise, material interface-wise, region interface-wise, and electrode-wise. The following example shows each of these possibilities:

```
LatticeHeatCapacity {
    cv = 1.1
}
Material = "Silicon" {
    LatticeHeatCapacity {
        cv = 1.63
    }
}
Region = "Oxide" {
    LatticeHeatCapacity {
        cv = 1.67
    }
}
MaterialInterface = "Silicon/Oxide" {
    LatticeHeatCapacity {
        cv = -1
    }
}
RegionInterface = "Oxide/Bulk" {
    LatticeHeatCapacity {
        cv = -2
    }
}
Electrode = "gate" {
    LatticeHeatCapacity {
        cv = -3
    }
}
```

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

```
    }  
}
```

As for the models themselves, not all parameters are valid in all locations, even though it is syntactically possible to specify them. For example, the heat capacity is not used for interfaces and electrodes. Therefore, it does not matter that the values provided in the example above are nonsensical.

To specify parameters for multiple parameter sets for the same location, put all these specifications together into one section for that location, as in the following example for dielectric permittivity and heat capacity:

```
Material = "Silicon" {  
    Epsilon{  
        epsilon= 11.6  
    }  
    LatticeHeatCapacity {  
        cv = 1.63  
    }  
}
```

Parameter files support an `Insert` statement to insert other parameter files. Inserted files themselves can use `Insert`. You can change parameters after an `Insert` statement. This is useful to provide standard values through the inserted file and to make the changes to those standards explicit. `Insert` is used as in the following example:

```
Material = "Silicon" { Insert = "Silicon.par" }
```

---

## Search Strategy for Parameter Files

Sentaurus Device uses the following strategy to search for inserted files, from highest to lowest priority as follows:

1. Local directory.
2. The `ParameterPath` variable in the `File` section can specify a list of releases. For example:

```
File {  
    ParameterPath = "2020.09 2021.06"  
}
```

In this case, the following directories are added to the search path:

```
$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB/2020.09  
$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB/2021.06
```

3. Sentaurus Device checks whether the environment variable `SDEVICEDB` is defined. This variable must contain a directory or a list of directories separated by white space or colons.

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

For example:

```
SDEVICEDEB= "/home/usr/lib /home/tcad/lib"
```

Sentaurus Device scans the directories in the given order until the inserted file is found.

**Note:**

The environment variable `SDEVICEDEB` also is used for the insert directive in Sentaurus Device command files (see [Inserting Files on page 189](#)).

#### 4. The default library directory:

```
$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB
```

In all cases, Sentaurus Device prints the path to the actual file and displays an error message if it cannot be found.

**Note:**

The insert directive is also available for command files (see [Inserting Files on page 189](#)).

---

## Parameters for Composition-Dependent Materials

The following parameter sets provide mole fraction dependencies. All models are available for compound semiconductors only, except where otherwise noted:

- Auger (recombination model)
- BalMob (mobility model)
- Band2BandTunneling
- Bandgap
- BandstructureParameters (see [Electronic Band Structure for Wurtzite Crystals on page 1082](#))
- Bennett (bandgap narrowing)
- ComplexRefractiveIndex
- ConstantMobility
- Deformation potential (elasticity modulus, the parameters for the electron band: `xis`, `dbs`, `xiu`, `xid` and, for hole band: `adp`, `bdp`, `ddp`, `dso`)
- deAlamo (bandgap narrowing)
- DirectTunneling (for semiconductor–insulator interfaces only)
- DopingDependence (mobility model)

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

- eDOSMass
- EnergyRelaxationTime
- Enormal (mobility model)
- Epsilon (also available for compound insulators)
- FEPolarization
- hDOSMass
- HighFieldDependence (mobility model)
- IALMob (mobility model)
- JainRoulston (bandgap narrowing)
- Kappa (lattice thermal conductivity, also available for compound insulators)
- LatticeHeatCapacity (also available for compound insulators)
- LatticeParameters
- MLDAQMModel
- NegInterfaceChargeMobility
- oldSlotboom (bandgap narrowing)
- PhuMob (mobility model)
- Piezoelectric\_Polarization
- PosInterfaceChargeMobility
- QuantumPotentialParameters
- QWStrain (see [Electronic Band Structure for Wurtzite Crystals on page 1082](#) and [Syntax for Quantum-Well Strain on page 1091](#))
- Radiative (recombination model)
- SchroedingerParameters
- SHEDistribution
- Slotboom (bandgap narrowing)
- SRH (recombination model)
- StressMobility (hSixBand model parameters)
- ThinLayerMobility

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

- `ToCurrentEnormal` (mobility model)
- `TransferredElectronEffect2` (mobility model)
- `vanOverstraetenDeMan` (impact ionization model)

Sentaurus Device supports the suppression of the mole fraction dependence of a given model and the use of a fixed (mole fraction-independent) parameter set instead. To suppress mole fraction dependency, specify the (fixed) values for the parameter (for example, `Eg0=1.53`) and omit all other coefficients associated with the interpolation over the mole fraction (for example, `Eg0(1)`, `B(Eg0(1))`, and `C(Eg0(1))`) from this section of the parameter file. It is not necessary to set them to zero individually.

#### Note:

When specifying a fixed value for one parameter of a given model, all other parameters for the same model must be fixed.

In summary, if the mole fraction dependence of a given model is suppressed, the parameter specification for this model is performed in exactly the same manner as for mole fraction-independent material.

## Ternary Semiconductor Composition

To illustrate a calculation of mole fraction-dependent parameter values for ternary materials, consider one mole interval from  $x_{i-1}$  to  $x_i$ .

For mole fraction value ( $x$ ) of this interval, to compute the parameter value ( $P$ ), Sentaurus Device uses the expression:

$$\begin{aligned} P &= P_{i-1} + A \cdot \Delta x + B_i \cdot \Delta x^2 + C_i \cdot \Delta x^3 \\ A &= \frac{\Delta P_i}{\Delta x_i} - B_i \cdot \Delta x_i - C_i \cdot \Delta x_i^2 \\ \Delta P_i &= P_i - P_{i-1} \\ \Delta x_i &= x_i - x_{i-1} \\ \Delta x &= x - x_{i-1} \end{aligned} \tag{1}$$

where  $P_i$ ,  $B_i$ ,  $C_i$ ,  $x_i$  are values defined in the parameter file for each mole fraction interval,  $x_0 = 0$ ,  $P_0$  is the parameter value (at  $x = 0$ ) specified using the same manner as for mole fraction-independent material (for example, `Eg0=1.53`). As in the formulas above, you are not required to specify coefficient  $A$  of the polynomial because it is easily recomputed inside Sentaurus Device.

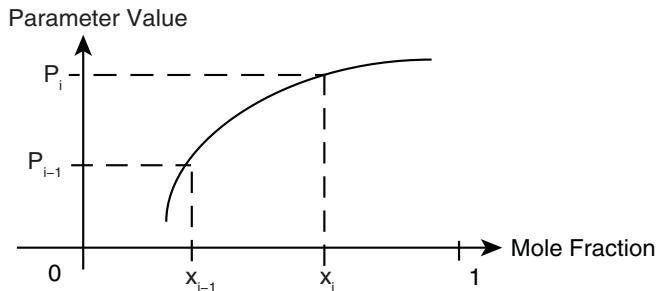
In the case of undefined parameters (these can be listed by printing the parameter file), Sentaurus Device uses linear interpolation using two parameter values of side materials (for  $x = 0$  and  $x = 1$ ):

$$P = (1-x)P_{x0} + xP_{x1} \tag{2}$$

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

**Figure 6** Parameter value as a function of mole fraction



### Example 1: Specifying Electric Permittivity

This example provides the specification of dielectric permittivity for  $\text{Al}_x\text{Ga}_{1-x}\text{As}$ :

```
Epsilon
{ * Ratio of the permittivities of material and vacuum
  epsilon = 13.18      # [1]
  * Mole fraction dependent model.
  * The linear interpolation is used on interval [0,1].
  epsilon(1) = 10.06    # [1]
}
```

A linear interpolation is used for the dielectric permittivity, where `epsilon` specifies the value for the mole fraction  $x = 0$ , and `epsilon(1)` specifies the value for  $x = 1$ .

### Example 2: Specifying Band Gap

This example provides a specification of the bandgap parameters for  $\text{Al}_x\text{Ga}_{1-x}\text{As}$ . A polynomial approximation, up to the third degree, describes the mole fraction-dependent band parameters on every mole fraction interval. In the following example, two intervals are used, namely,  $[\text{xmax}(0), \text{xmax}(1)]$  and  $[\text{xmax}(1), \text{xmax}(2)]$ . The parameters `Eg0`, `Chi0`, ... correspond to the values for  $X = \text{xmax}(0)$ ; while the parameters `Eg0(1)`, `Chi0(1)`, ... correspond to the values for  $X = \text{xmax}(1)$  and, finally, `Eg0(2)`, `Chi0(2)`, ... correspond to the values for  $X = \text{xmax}(2)$ .

The coefficients  $A$  and  $F$  of the following polynomial are determined from the values at both ends of the intervals:

$$F + A(X - \text{xmin}(I)) + B(X - \text{xmin}(I))^2 + C(X - \text{xmin}(I))^3 \quad (3)$$

The coefficients  $B$  and  $C$  must be specified explicitly. You can introduce additional intervals:

```
Bandgap *temperature dependent*
{ * Eg = Eg0 - alpha T^2 / (beta + T) + alpha Tpar^2 / (beta + Tpar)
  * Eg0 can be overwritten in below bandgap narrowing models,
  * if any of the BGN model is chosen in physics section.
  * Parameter 'Tpar' specifies the value of lattice
  * temperature, at which parameters below are defined.
  Eg0 = 1.42248      # [ev]
```

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

```
Chi0 = 4.11826      # [eV]
alpha = 5.4050e-04  # [eV K^-1]
beta = 2.0400e+02   # [K]
Tpar = 3.0000e+02   # [K]
* Mole fraction dependent model.
* The following interpolation polynomial can be used on
* interval [Xmin(I),Xmax(I)]:
* F(X) = F(I-1)+A(I)*(X-Xmin(I))+B(I)*(X-Xmin(I))
*           ^2+C(I)*(X-Xmin(I))^3,
* where Xmax(I), F(I), B(I), C(I) are defined below for each interval.
* A(I) is calculated for a boundary condition F(Xmax(I)) = F(I).
* Above parameters define values at the following mole fraction:
    Xmax(0) = 0.0000e+00  # [1]
* Definition of mole fraction intervals, parameters, and coefficients:
    Xmax(1) = 0.45          # [1]
    Eg0(1) = 1.98515        # [eV]
    B(Eg0(1)) = 0.0000e+00  # [eV]
    C(Eg0(1)) = 0.0000e+00  # [eV]
    Chi0(1) = 3.575         # [eV]
    B(Chi0(1)) = 0.0000e+00 # [eV]
    C(Chi0(1)) = 0.0000e+00 # [eV]
    alpha(1) = 4.7727e-04   # [eV K^-1]
    B(alpha(1)) = 0.0000e+00 # [eV K^-1]
    C(alpha(1)) = 0.0000e+00 # [eV K^-1]
    beta(1) = 1.1220e+02    # [K]
    B(beta(1)) = 0.0000e+00 # [K]
    C(beta(1)) = 0.0000e+00 # [K]
    Xmax(2) = 1              # [1]
    Eg0(2) = 2.23            # [eV]
    B(Eg0(2)) = 0.143        # [eV]
    C(Eg0(2)) = 0.0000e+00  # [eV]
    Chi0(2) = 3.5             # [eV]
    B(Chi0(2)) = 0.0000e+00 # [eV]
    C(Chi0(2)) = 0.0000e+00 # [eV]
    alpha(2) = 4.0000e-04    # [eV K^-1]
    B(alpha(2)) = 0.0000e+00 # [eV K^-1]
    C(alpha(2)) = 0.0000e+00 # [eV K^-1]
    beta(2) = 0.0000e+00     # [K]
    B(beta(2)) = 0.0000e+00 # [K]
    C(beta(2)) = 0.0000e+00 # [K]
}
```

## Quaternary Semiconductor Composition

Sentaurus Device supports 1:3, 2:2, and 3:1 III–V quaternary alloys. A 1:3 III–V quaternary alloy is given by:

$$AB_xC_yD_z \quad (4)$$

where  $A$  is a group III element, and  $B$ ,  $C$ , and  $D$  are group V elements (usually listed according to increasing atomic number).

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

Conversely, a 3:1 III–V quaternary alloy can be described as:

$$A_x B_y C_z D \quad (5)$$

where  $A$ ,  $B$ , and  $C$  are group III elements, and  $D$  is a group V element.

The composition variables  $x$ ,  $y$ , and  $z$  are nonnegative, and they are constrained by:

$$x + y + z = 1 \quad (6)$$

An example would be  $\text{Al}_x\text{Ga}_y\text{In}_{1-x-y}\text{As}$ , where  $1 - x - y$  corresponds to  $z$ .

Sentaurus Device uses the symmetric interpolation scheme proposed by Williams *et al.* [1] to compute the parameter value  $P(A_x B_y C_z D)$  of a 3:1 III–V quaternary alloy as a weighted sum of the corresponding ternary values:

$$P(A_x B_y C_z D) = \frac{xyP(A_{1-u}B_uD) + yzP(B_{1-v}C_vD) + xzP(A_{1-w}C_wD)}{xy + yz + xz} \quad (7)$$

where:

$$u = \frac{1-x+y}{2}, v = \frac{1-y+z}{2}, w = \frac{1-x+z}{2} \quad (8)$$

The parameter values  $P(AB_xC_yD_z)$  for 1:3 III–V quaternary alloys are computed similarly. A general 2:2 III–V quaternary alloy is given by:

$$A_x B_{1-x} C_y D_{1-y} \quad (9)$$

where  $A$  and  $B$  are group III elements, and  $C$  and  $D$  are group V elements. The composition variables  $x$  and  $y$  satisfy the inequalities  $0 \leq x \leq 1$  and  $0 \leq y \leq 1$ . As an example, the material  $\text{In}_{1-x}\text{Ga}_x\text{As}_y\text{P}_{1-y}$  is mentioned.

The parameters  $P(A_x B_{1-x} C_y D_{1-y})$  of a 2:2 III–V quaternary alloy are determined by interpolation between the four ternary side materials:

$$P(A_x B_{1-x} C_y D_{1-y}) = \frac{x(1-x)(yP(A_x B_{1-x} C) + (1-y)P(A_x B_{1-x} D)) + y(1-y)(xP(A C_y D_{1-y}))}{x(1-x) + y(1-y)} + \frac{(1-x)P(B C_y D_{1-y})}{x(1-x) + y(1-y)} \quad (10)$$

The interpolation of model parameters for quaternary alloys is also discussed in the literature [2][3][4][5]. A comprehensive survey paper is available [6].

## Default Model Parameters for Compound Semiconductors

It is important to understand how the default values for different physical models in different materials are determined. The approach used in Sentaurus Device is summarized here. For example, consider the material `Material1`. Assume that no default parameters are defined for this material and a given physical model `Model1`.

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

In this case, use the command `sdevice -P:Material` to see for which models specific default parameters are predefined in the material `Material`:

1. Silicon parameters are used, by default, in the model `Model` if the material `Material` is mole fraction independent.
2. If `Material` is a compound material and dependent on the mole fraction  $x$ , the default values of the parameters for the model `Model` are determined by a linear interpolation between the values of the respective parameters of the corresponding ‘pure’ materials (that is, materials corresponding to  $x = 0$  and  $x = 1$ ). For example, for  $\text{Al}_x\text{Ga}_{1-x}\text{As}$ , values of the parameters of GaAs and AlAs are used in the interpolation formula.
3. If `Material` is a quaternary material and dependent on  $x$  and  $y$ , an interpolation formula, which is based on the values of all corresponding ternary materials, is used. For example, for InGaAsP, the values of four materials (InAsP, GaAsP, GaInP, and InGaAs) are used in the interpolation procedure to obtain the default values of the parameters.

Additional details for each model and specific materials are found in the comments of the parameter file.

If you provide an inadequate mole fraction range in the parameter file, a warning message is displayed to notify you about the mole fraction-dependent quantity and region where you might want to correct the mole fraction range. For example, if the compound semiconductor material in a TDR structure has mole fraction ranges from 0 to 0.6, but you supply the mole fraction range from 0 to 0.59 for a mole fraction-dependent quantity in the parameter file, then a warning message is displayed.

---

## Combining Parameter Specifications

Sentaurus Device has built-in values for many parameters, can read parameters from files in a default location, and can read a user-supplied parameter file that can specify parameters for various locations. This section discusses the rules for combining all these specifications.

Parameter handling is affected by the presence of the flag `DefaultParametersFromFile` (specified in the global `Physics` section) and the value of `ParameterInheritance` (specified in the global `Math` section). By default, `ParameterInheritance=Flatten`; by setting `ParameterInheritance=None`, the rules for combining parameter specifications can be altered.

## Materialwise Parameters

To determine materialwise parameters, follow these steps:

1. The parameters are initialized from built-in values. These values can depend on the material. For many materials, no appropriate built-in values exist, and silicon values are used instead.
2. If the `DefaultParametersFromFile` flag is present, then Sentaurus Device reads the default parameter file for the material. This file can add parameters (for example, add intervals to a mole fraction specification) or overwrite built-in values (for details, see [Default Parameters on page 84](#)).
3. If `ParameterInheritance=Flatten` and specifications outside any location-specific section are present, then they can add or overwrite the parameter values obtained through the previous two steps, and they also contribute to a separate, global, default parameter set.
4. If your parameter file contains a section for the material, then specifications in this section can add to or overwrite the parameter values obtained through the previous three steps. If `ParameterInheritance=None`, specifications outside any location-specific section are handled as if they occurred in a section for the material silicon.

Materialwise specifications in your parameter file are read even when the material is not contained in the structure to be simulated. This is useful, for example, to provide parameters for corner and side materials for materials that are present in the structure.

Apart from corner and side materials, materialwise parameters are rarely used directly; physical bulk models access parameters regionwise, not materialwise. However, materialwise settings can affect regionwise parameters. The next section explains this in detail.

## Regionwise Parameters

When your parameter file does not contain a section for a certain region, the parameter values for this region are the same as for the material of the region. In this case, if `ParameterInheritance=None`, the regionwise parameters are discarded entirely and the material parameters are accessed instead. If `ParameterInheritance=Flatten`, the region parameters are a copy of the material parameters (the subtle distinction of these two cases becomes important only when ramping parameters).

If your parameter file does contain a section for a certain region, the parameters for this region are initialized executing Steps 1 and 2 as for the materialwise parameters, using the material of the region to select the built-in values and the default parameter file. If `ParameterInheritance=Flatten`, and a section for this material is present in your parameter file, Steps 3 and 4 are taken as well. Finally, in any case, the section for the region is evaluated and can add to or overwrite the values previously obtained.

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

#### Note:

If `ParameterInheritance=None` and a section for a region is present in your parameter file, then none of the settings for any parameter in the section for the material of the region (if such a section exists) has an effect on the parameters for that region. In other words, if `ParameterInheritance=None`, then the region parameters do not ‘inherit’ user settings from the material parameters.

## Material Interface–Wise Parameters

Material interface–wise parameters are handled similarly to materialwise parameters, simply replace the term ‘material’ by ‘material interface’ in the description of the handling of materialwise parameters.

## Region Interface–Wise Parameters

Region interface–wise parameters are handled similarly to regionwise parameters, and the relation between region interface–wise parameters and material interface–wise parameters is analogous to the relation between regionwise and materialwise parameters. As an additional complication, if `ParameterInheritance=None` and neither a section for the region interface nor for its material interface is present in your parameter file, the region interface parameters are discarded, and the parameters for material silicon are accessed instead.

## Electrode-Wise Parameters

Electrode-wise parameters are handled similarly to materialwise parameters, simply replace the term ‘material’ by ‘electrode’ in the description of the handling of materialwise parameters.

---

## Generating a Copy of Parameter File

To redefine a parameter value for a particular material, a copy of the default parameter file must be created. To do this, the command `sdevice -P` prints the parameter file for silicon, with insulator properties. [Table 3](#) lists the principal options for the command `sdevice -P`.

*Table 3 Principal options for generating a parameter file*

Option	Description
<code>-P:All</code>	Prints a copy of the parameter file for all materials. Materials are taken from the file <code>datexcodes.txt</code> .
<code>-P:Material</code>	Prints model parameters for the specified material.

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

Table 3 Principal options for generating a parameter file (Continued)

Option	Description
-P:Material:x	Prints model parameters for the specified material for mole fraction $x$ .
-P:Material:x:y	Prints model parameters for the specified material for mole fractions $x$ and $y$ .
-P filename	Prints model parameters for materials and interfaces used in the command file <code>filename</code> .
-r	Reads parameters from default locations (see <a href="#">Search Strategy for Parameter Files on page 71</a> ) before printing parameters. This is analogous to using the <code>DefaultParametersFromFile</code> flag (see <a href="#">Materialwise Parameters on page 79</a> ).

For regionwise and materialwise parameter specifications, any model and parameter from the default section is usable, even if it is not printed for the particular material.

For mole fraction-dependent parameters, for people, it is difficult to read a parameter file and to obtain the final values of parameters (for example, the band gap) for a particular composition mole fraction. By using the command `sdevice -M <inputfile.cmd>`, Sentaurus Device creates a `models-M.par` file that will contain regionwise parameters with only constant values (instead of the polynomial coefficients) for regions where the composition mole fraction is constant. For regions where the composition is not a constant, Sentaurus Device prints the default material parameters.

As an alternative to the command `sdevice -P`, Sentaurus Device provides the `sdevice -L` command, which supports the same options in [Table 3](#) for `sdevice -P`. However, instead of generating a single file, it creates separate files for each material and each material interface. Typically, these files are edited and used to provide default parameters (see [Default Parameters on page 84](#)).

For example, assume the command file `ppl_des.cmd` is for a simple silicon MOSFET with four electrodes, one silicon region, and one oxide region. Using the following command, a parameter file is created in the current directory for each material, material interface, and electrode found in `nmos_mdr.tdr`:

```
sdevice -L ppl_des.cmd
```

In this example, the appropriate files are `Silicon.par`, `Oxide.par`, `Oxide%Silicon.par`, and `Electrode.par`.

In addition, the following `models.par` file is created in the current directory:

```
Region = "Region0" { Insert = "Silicon.par" }
Region = "Region1" { Insert = "Oxide.par" }
RegionInterface = "Region1/Region0" { Insert = "Oxide%Silicon.par" }
```

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

```
Electrode = "gate" { Insert = "Electrode.par" }
Electrode = "source" { Insert = "Electrode.par" }
Electrode = "drain" { Insert = "Electrode.par" }
Electrode = "substrate" { Insert = "Electrode.par" }
```

This `models.par` file can be renamed. It is only used in the simulation if it is specified in the `File` section of the command file of Sentaurus Device:

```
File {...  
Parameter = "models.par"  
...}
```

---

## Undefined Physical Models

For a nonsilicon simulation, the default behavior of Sentaurus Device is to use silicon parameters for models that are not defined in a material used in the simulation. This is also the case if a material parameter is undefined for a certain mole fraction range and the device uses a mole fraction in this undefined range. It is useful for noncritical models, but it can lead to confusion, for example, if the semiconductor band gap is not defined, and Sentaurus Device uses that of silicon. Therefore, Sentaurus Device has a list of critical models and stops the simulation, with an error message, if these models are not defined.

**Note:**

The model is defined in a material if it is present in the default parameter file of Sentaurus Device for the material or it is specified in a user-defined parameter file.

**Table 4** lists the critical models (with names from the parameter file of Sentaurus Device) with materials where these models are checked.

*Table 4 List of critical models*

Model	Insulator	Semiconductor	Conductor
Auger		x	
Bandgap		x	
ConstantMobility		x	
DopingDependence		x	
eDOSmass hDOSmass		x	
Epsilon	x	x	

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

Table 4     *List of critical models (Continued)*

Model	Insulator	Semiconductor	Conductor
Kappa	x	x	x
RadiativeRecombination		x	
RefractiveIndex	x	x	
Scharfetter		x	
SchroedingerParameters	x	x	

The models `Bandgap`, `DOSmass`, and `Epsilon` are checked always. For other models, this check is performed for each region, but only if appropriate models in the `Physics` section and equations in the `Solve` section are activated. The thermal conductivity model `Kappa` is checked only if the lattice temperature equation is included.

Drift-diffusion or hydrodynamic simulations activate the checking mobility models `ConstantMobility` and `DopingDependence` (with appropriate models in the `Physics` section).

#### Note:

This checking procedure can be switched off by the `-CheckUndefinedModels` keyword in the `Math` section.

The models `eDOSmass` and `hDOSmass` also must be defined for insulators to support tunneling (see [Nonlocal Tunneling Parameters on page 830](#)). Only the following specifications are acceptable for insulators:

```
eDOSmass {  
    Formula = 1  
    a = 0  
    m1 = 0  
    mm = <effective mass>      # default 0.42  
}  
  
hDOSmass {  
    Formula = 1  
    a = 0  
    b = 0  
    c = 0  
    d = 0  
    e = 0  
    f = 0  
    g = 0  
    h = 0
```

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

```
i = 0
mm = <effective mass>      # default 1
}
```

#### Note:

In general, the models `eDOSmass` and `hDOSmass` only need to be specified for user-specified insulators. The correct values are predefined for standard insulators.

---

## Default Parameters

Sentaurus Device provides built-in default parameters for many models and materials. However, Sentaurus Device also offers the option to overwrite the built-in values with default parameters from files. This option is activated by `DefaultParametersFromFile` in the global `Physics` section:

```
Physics {
    DefaultParametersFromFile
    ...
}
```

Table 5     *File names that are used to initialize default parameters*

Location of parameters	File name
Materials	<material>.par, for example, Silicon.par, GaAs.par
Material interfaces	<material1>%<material2>.par, for example, InAlAs%AlGaAs.par (Instead of %, a comma or space can also be used.)
Contacts	Contact.par or Electrode.par

Sentaurus Device uses the same search strategy as for inserted files (see [Physical Model Parameters on page 70](#)). Sentaurus Device ships with a selection of parameter files in the directory `$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB`.

This directory also contains release-specific subdirectories. For example:

```
$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB/2020.09
$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB/2021.06
```

You can request that the parameter files from a specific release are taken by specifying the `ParameterPath` variable in the `File` section:

```
File {
    ParameterPath = "2021.06"
}
```

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

If a matching file is not found, then the built-in default parameters are used unaltered. Check the log file of Sentaurus Device to see which files were actually used.

---

## Named Parameter Sets

Some models in Sentaurus Device support the use of parameter sets that can be named. For example, `EnormalDependence` is the unnamed parameter set used with the Lombardi mobility model. In the parameter file, you can write the following to declare a parameter set for the Lombardi mobility model with the name `myset`:

```
EnormalDependence "myset" {
    B = 3.6100e+07 , 1.5100e+07      # [cm/s]
    C = 1.7000e+04 , 4.1800e+03      # [cm^(5/3)/(V^(2/3)s)]
    ...
}
```

Typically, named parameter sets are used to store alternative parameterizations for a model. They can be selected from the command file by specifying the name with `ParameterSetName` as an option to a model that supports this feature. For example:

```
Physics {
    Mobility (
        PhuMob
        Enormal( Lombardi (ParameterSetName="myset" ) )
        HighFieldSaturation
    )
}
```

If `ParameterSetName` is not specified, the unnamed parameter set associated with the model is used by default.

*Table 6 Models that support named parameter sets*

Model name	Parameter set (unnamed)
eQuantumPotential	QuantumPotentialParameters
hQuantumPotential	

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

*Table 6 Models that support named parameter sets (Continued)*

Model name		Parameter set (unnamed)
Mobility eMobility hMobility	Enormal(Lombardi) ToCurrentEnormal(Lombardi)	EnormalDependence
	ThinLayer(Lombardi)	EnormalDependence ThinLayerMobility
	Enormal(IALMob) ToCurrentEnormal(IALMob)	IALMob
	ThinLayer(IALMob)	IALMob ThinLayerMobility
	HighFieldSaturation eHighFieldSaturation hHighFieldSaturation Diffusivity eDiffusivity hDiffusivity	HighFieldDependence HydroHighFieldDependence
	Tensor eTensor hTensor	Piezoresistance
Piezo(Mobility)	Factor eFactor hFactor	Piezoresistance EffectiveStressModel
	Recombination(Band2Band)	Band2BandTunneling

---

## Auto-Orientation Framework

Some models in Sentaurus Device support an auto-orientation framework that automatically switches between different named parameter sets for model evaluation, based on the surface orientation of the nearest interface.

When the `AutoOrientation` option is activated for a model, by default, Sentaurus Device looks for and uses parameter sets named "100", "110", and "111" corresponding to surface orientations of {100}, {110}, and {111}. If the nearest interface orientation is {110}, for example, the model is evaluated using the parameter set "110". For surface orientations

## Chapter 2: Specifying Physical Devices

### Physical Model Parameters

that fall between {100}, {110}, and {111}, the named parameter set that most closely corresponds to the actual surface orientation is used.

Models that support the auto-orientation framework include:

- Density-gradient quantization model (`eQuantumPotential`, `hQuantumPotential`)
- Lombardi and IALMob mobility models
- HighFieldSaturation and Diffusivity
- ThinLayer mobility model
- Piezoresistance models (`Tensor`, `eTensor`, `hTensor`, `Factor`, `eFactor`, `hFactor`)
- EffectiveStressModel (`Factor`, `eFactor`, `hFactor`)

In the `Plot` section of the command file, the quantities `InterfaceOrientation` and `NearestInterfaceOrientation` can be specified to see the orientation that is used, at each interface face vertex and at every vertex, respectively, when auto-orientation is enabled for a model.

## Changing Orientations Used With Auto-Orientation

The auto-orientation framework can be modified to use surface orientations other than {100}, {110}, and {111}. This is accomplished by providing a definition for `AutoOrientation` in the `Math` section of the command file:

```
Math { AutoOrientation=(ori1, ori2, ...) }
```

In this specification, the `orii` values are three-digit integers that represent Miller indices for families of equivalent planes (a cubic lattice structure is assumed). For example, to modify `AutoOrientation` to use the surface orientations {100}, {110}, {111}, and {211}, specify:

```
Math { AutoOrientation=(100, 110, 111, 211) }
```

In this case, models that support `AutoOrientation` will switch between the parameter sets named "100", "110", "111", and "211", depending on the surface orientation of the nearest interface.

## Auto-Orientation Smoothing

By default, the switch from one parameter set to another when using auto-orientation occurs abruptly. To enable a smooth transition between different parameter sets, specify a nonzero value for the auto-orientation smoothing distance in the `Math` section:

```
Math {
    AutoOrientationSmoothingDistance = 0.005    # [micrometers]
}
```

## Chapter 2: Specifying Physical Devices

### References

The smoothing distance is an approximate measure of the distance over which the transition between different parameter sets will occur. For convenience, specifying `AutoOrientationSmoothingDistance < 0` uses the average interface vertex spacing as the smoothing distance. In many cases, this is a reasonable choice.

When auto-orientation smoothing is used, weights are calculated at each vertex for each orientation-dependent parameter set based on the proximity to the different supported surface orientations:

$$w_i(\text{vertex}) = \sum_{j=1}^{N_i} \frac{A_j}{A_{\text{total}}} \exp\left(-\frac{d_j - d_{\min}}{d_{\text{smooth}}}\right), i = 100, 110, \dots \quad (11)$$

The summation is over all interface vertices associated with orientation  $i$ :

- $d_{\min}$  is the minimum distance to the interface.
- $d_j$  is the distance to the interface vertex  $j$ .
- $d_{\text{smooth}}$  is the smoothing distance.
- $A_j$  is the area associated with the interface vertex  $j$ .
- $A_{\text{total}}$  is the total interface area.

At each vertex, very small weights are set to zero, and the remaining weights are normalized so that their sum is equal to one.

During model evaluation, the weights for each orientation at a vertex are used to obtain a weighted average of the quantity being calculated.

In the `Plot` section of the command file, the quantity `AutoOrientationSmoothing` can be specified to see the vertices where auto-orientation smoothing is used and the dominant orientation at those vertices.

---

## References

- [1] C. K. Williams *et al.*, “Energy Bandgap and Lattice Constant Contours of III-V Quaternary Alloys of the Form  $A_xB_yC_zD$  or  $AB_xC_yD_z$ ,” *Journal of Electronic Materials*, vol. 7, no. 5, pp. 639–646, 1978.
- [2] T. H. Glisson *et al.*, “Energy Bandgap and Lattice Constant Contours of III-V Quaternary Alloys,” *Journal of Electronic Materials*, vol. 7, no. 1, pp. 1–16, 1978.
- [3] M. P. C. M. Krijn, “Heterojunction band offsets and effective masses in III–V quaternary alloys,” *Semiconductor Science and Technology*, vol. 6, no. 1, pp. 27–31, 1991.

## Chapter 2: Specifying Physical Devices

### References

- [4] S. Adachi, "Band gaps and refractive indices of AlGaAsSb, GaInAsSb, and InPAsSb: Key properties for a variety of the 2–4-mm optoelectronic device applications," *Journal of Applied Physics*, vol. 61, no. 10, pp. 4869–4876, 1987.
- [5] R. L. Moon, G. A. Antypas, and L. W. James, "Bandgap and Lattice Constant of GaInAsP as a Function of Alloy Composition," *Journal of Electronic Materials*, vol. 3, no. 3, pp. 635–644, 1974.
- [6] I. Vurgaftman, J. R. Meyer, and L. R. Ram-Mohan, "Band parameters for III–V compound semiconductors and their alloys," *Journal of Applied Physics*, vol. 89, no. 11, pp. 5815–5875, 2001.

# 3

## Mixed-Mode Simulations

---

*This chapter describes how to specify circuits and compact devices for mixed-mode simulations.*

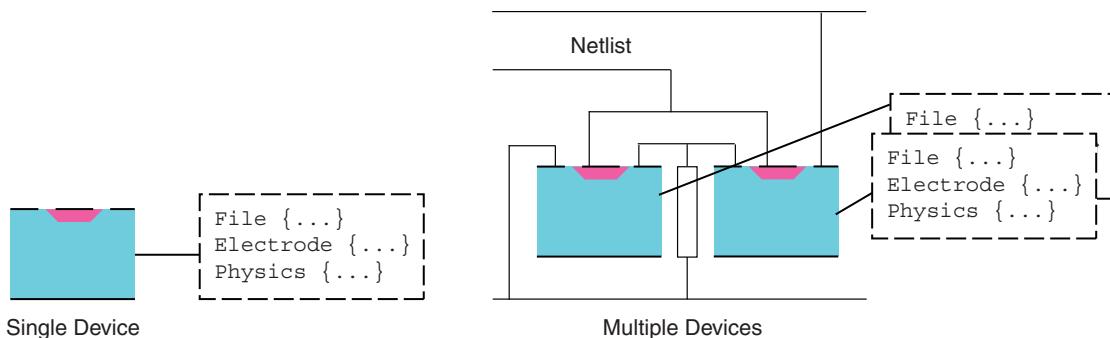
---

### Overview of Mixed-Mode Simulations

Sentaurus Device is a single-device simulator, and a mixed-mode device and circuit simulator. A single-device command file is defined through the mesh, contacts, physical models, and solve command specifications.

For a multidevice simulation, the command file must include specifications of the mesh (`File` section), contacts (`Electrode` section), and physical models (`Physics` section) for each device. A circuit netlist must be defined to connect the devices (see [Figure 7](#)), and solve commands must be specified that solve the whole system of devices.

*Figure 7     Each device in a multidevice simulation is connected with a circuit netlist*



The `Device` command defines each physical device. A command file can have any number of `Device` sections. The `Device` section defines a device, but a `System` section is required to create and connect devices.

## Chapter 3: Mixed-Mode Simulations

### Compact Models

Sentaurus Device provides different compact models for use in mixed-mode simulations and supports different ways of specifying compact models as follows:

- Netlist files support a subset of the Synopsys PrimeSim™ HSPICE® language. You can use these files to specify parameter sets and instances of SPICE models and PrimeSim HSPICE models (see [PrimeSim HSPICE Netlist Files on page 95](#)).
- SPICE circuit files (extension `.scf`) are available for SPICE models, PrimeSim HSPICE models, and built-in models. Both parameter sets and instances can be specified (see [SPICE Circuit Files on page 101](#)).
- Compact circuit files (extension `.ccf`) support the compact model interface in Sentaurus Device. Devices and parameter sets for user-defined models must be specified in compact circuit files. Instances can appear either in compact circuit files or in the `System` section of the command file (see [User-Defined Compact Models on page 94](#)).
- Electrical and thermal circuits can be defined in the `System` section of the command file. Initial conditions for the circuit nodes can be specified, as well as `Plot` statements to generate output (see [System Section on page 103](#)).

---

## Compact Models

Sentaurus Device provides different compact models for use in mixed-mode simulations:

- SPICE models (see [Table 7 on page 92](#))
- Frequently used PrimeSim HSPICE models (see [Table 8 on page 93](#))
- Built-in, special-purpose models (see [Table 9 on page 94](#))
- User-defined models (see [User-Defined Compact Models on page 94](#))

This section describes the incorporation of compact models in mixed-mode simulations. The *Compact Models User Guide* provides references for the different models.

---

## Hierarchical Description of Compact Models

In Sentaurus Device, each compact model is described by a three-level hierarchy:

- **Device:** The device describes the basic properties of a compact model. This includes the names of the model, electrodes, thermodes, and internal variables, and the names and types of the internal states and parameters. Devices are predefined for SPICE models and built-in models. For user-defined models, you must specify the devices.
- **Parameter set:** Each parameter set is derived from a device. It defines default values for the parameters of a compact model. Usually, the parameters that are shared among

## Chapter 3: Mixed-Mode Simulations

### Compact Models

several instances are defined in a parameter set. Most SPICE models and built-in models provide a default parameter set, which can be directly referenced in a circuit description. For more complicated models such as MOSFETs, you can introduce new parameter sets.

- **Instance:** Instances correspond to the elements in the Sentaurus Device circuit. Each instance is derived from a parameter set. It can override the values of its parameters if necessary.

For SPICE models and built-in models, you define parameter sets and instances. For user-defined models, it is possible (and required) to introduce new devices. For more information, see the *Compact Models User Guide*.

Table 7 Available SPICE models in Sentaurus Device

Model	Device name	Name of default parameter set
Resistor	Resistor	Resistor_pset
Capacitor	Capacitor	Capacitor_pset
Inductor	Inductor	Inductor_pset
Coupled inductors	mutual	mutual_pset
Voltage-controlled switch	Switch	Switch_pset
Current-controlled switch	cswitch	cswitch_pset
Voltage source	Vsource	Vsource_pset
Current source	Isource	Isource_pset
Voltage-controlled current source	VCCS	VCCS_pset
Voltage-controlled voltage source	VCVS	VCVS_pset
Current-controlled current source	CCCS	CCCS_pset
Current-controlled voltage source	CCVS	CCVS_pset

### Chapter 3: Mixed-Mode Simulations

#### Compact Models

*Table 7 Available SPICE models in Sentaurus Device (Continued)*

Model	Device name	Name of default parameter set
MOSFET	Mos1	Mos1_pset
	Mos2	Mos2_pset
	Mos3	Mos3_pset
	Mos6	Mos6_pset
	BSIM1	BSIM1_pset
	BSIM2	BSIM2_pset
	BSIM3	BSIM3_pset
	BSIM4	BSIM4_pset
	B3SOI	B3SOI_pset
Diode	Diode	Diode_pset
Bipolar junction transistor	BJT	BJT_pset
Junction field effect transistor	JFET	JFET_pset
GaAs MESFET	MES	MES_pset

*Table 8 Available PrimeSim HSPICE models in Sentaurus Device*

Model	Device Name	Model	Device Name
HSPICE Level 1	HMOS_L1	HSPICE Level 61	HMOS_L61
HSPICE Level 2	HMOS_L2	HSPICE Level 62	HMOS_L62
HSPICE Level 3	HMOS_L3	HSPICE Level 64	HMOS_L64
HSPICE Level 28	HMOS_L28	HSPICE Level 68	HMOS_L68
HSPICE Level 49	HMOS_L49	HSPICE Level 69	HMOS_L69
HSPICE Level 53	HMOS_L53	HSPICE Level 72	HMOS_L72

## Chapter 3: Mixed-Mode Simulations

### Compact Models

Table 8 Available PrimeSim HSPICE models in Sentaurus Device (Continued)

Model	Device Name	Model	Device Name
HSPICE Level 54	HMOS_L54	HSPICE Level 73	HMOS_L73
HSPICE Level 57	HMOS_L57	HSPICE Level 76	HMOS_L76
HSPICE Level 59	HMOS_L59		

Table 9 Built-in models in Sentaurus Device

Model	Device name	Name of default parameter set
Electrothermal resistor	Ter	Ter_pset
Ferroelectric capacitor	ferroelectric	ferroelectric_pset
MOS harness	MOS_harness	MOS_harness_pset
Parameter interface	Param_Interface_Device	Param_Interface
Saturable inductor	satinductor	satinductor_pset
SPICE temperature interface	Spice_Temperature_Interface_Device	Spice_Temperature_Interface

## User-Defined Compact Models

Sentaurus Device provides a compact model interface (CMI) for user-defined compact models. These models are implemented in C++ by you and linked to Sentaurus Device at runtime. Access to the source code of Sentaurus Device is not required.

To implement a new user-defined model:

1. Provide a corresponding equation for each variable in the compact model. For electrode voltages, compute the current flowing from the device into the electrode. For an internal model variable, use a model equation.
2. Write a formal description of the new compact model. Sentaurus Device reads this compact circuit file before the model is loaded.
3. Implement a set of C++ interface subroutines. Sentaurus Device provides a runtime environment.

## Chapter 3: Mixed-Mode Simulations

### PrimeSim HSPICE Netlist Files

4. Compile the model code into a shared object file, which is linked at runtime to Sentaurus Device. A `cmi` script executes this compilation.
5. Use the `CMPPath` keyword to define a search path in the `File` section of the command file.
6. Reference user-defined compact models in compact circuit files (extension `.ccf`) or directly in the `System` section of the command file.

---

## PrimeSim HSPICE Netlist Files

Sentaurus Device accepts PrimeSim HSPICE netlists that are provided in a separate file. A netlist is specified in the `System` section as follows:

```
System { Netlist = "spice_netlist.sp" }
```

Sentaurus Device can process only a subset of the PrimeSim HSPICE netlist syntax.

The recognized syntax is described in the following sections.

---

## Structure of Netlist Files

The first line of a netlist file is assumed to be a title line and is ignored. For example:

```
.TITLE 'amplifier netlist'
```

The title line is followed by a sequence of PrimeSim HSPICE statements, and the netlist is terminated by an optional `.END` statement:

```
.END
```

Everything after the final `.END` statement is ignored.

The netlist parser is case insensitive, except for string literals or file names in `.INCLUDE` statements:

```
.PARAM s = str('This is a case sensitive string.')
.INCLUDE 'Case/Sensitive/Filename'
```

## Comments

A line starting with either the dollar sign (\$) character or the asterisk (\*) character is a comment line. For example:

```
* This is a comment.
```

You can use in-line comments after the \$ character:

```
R1 1 2 R=100 $ drain resistor
```

## Chapter 3: Mixed-Mode Simulations

PrimeSim HSPICE Netlist Files

## Continuation Lines

Use the plus sign (+) character in the first column to indicate a continuation line:

```
R1 1 0  
+ R=500
```

## The .INCLUDE Statement

Use the .INCLUDE statement to include another netlist in the current netlist:

```
.INCLUDE models.sp
```

## Numeric Constants

You can enter numbers in one of the following formats:

- Integer (for example, 7)
- Floating point (for example, -4.5)
- Floating point with an integer exponent (for example, 3e8 and -1.2e9)
- Integer with a scale factor listed in [Table 10](#) (for example, 6k)
- Floating point with a scale factor listed in [Table 10](#) (for example, -8.9meg)

### Note:

The scale factor *a* is not a scale factor in a character string that contains amps. For example, the expression 20amps is interpreted as 20 amperes of current, not as 20e-18mps.

*Table 10 Scale factors*

Scale factor	Description	Multiplying factor
t	tera	$10^{12}$
g	giga	$10^9$
meg or x	mega	$10^6$
k	kilo	$10^3$
mil	one-thousandth of an inch	$25.4 \cdot 10^{-6}$
m	milli	$10^{-3}$

## Chapter 3: Mixed-Mode Simulations

PrimeSim HSPICE Netlist Files

Table 10 Scale factors (Continued)

Scale factor	Description	Multiplying factor
u	micro	$10^{-6}$
n	nano	$10^{-9}$
p	pico	$10^{-12}$
f	femto	$10^{-15}$
a	atto	$10^{-18}$

## Parameters and Expressions

In the PrimeSim HSPICE tool, parameters are names that you associate with a value. Numeric and string parameters are supported. For example:

```
.PARAM a = 4
.PARAM b = '2*a + 7'
.PARAM s = str('This is a string')
.PARAM t = str(s)
```

The following built-in mathematical functions are supported:

sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, abs, sqrt, pow, pwr, log, log10, exp, db, int, nint, sgn, sign, floor, ceil, min, max

## Subcircuits

Reusable cells can be specified as subcircuits. The general definition is given by:

```
.SUBCKT name n1 n2 ... [param1=val] [param2=val] ...
.ENDS
```

or:

```
.MACRO name n1 n2 ... [param1=val] [param2=val] ...
.EOM
```

String parameters are supported as well:

```
.SUBCKT name n1 n2 ... [param=str('string')] ...
.ENDS
```

## Chapter 3: Mixed-Mode Simulations

PrimeSim HSPICE Netlist Files

### Examples

```
.PARAM P5=5 P2=10

.SUBCKT SUB1 1 2 P4=4
    R1 1 0 P4
    R2 2 0 P5
    X1 1 2 SUB2 P6=7
    X2 1 2 SUB2
.ENDS

.MACRO SUB2 1 2 P6=11
    R1 1 2 P6
    R2 2 0 P2
.EOM

X1 1 2 SUB1 P4=6

X2 3 4 SUB2 P6=15
```

### The .MODEL Statement

The .MODEL statement has the following general syntax:

```
.MODEL model_name type [level=num] [pname1=val1] [pname2=val2] ...
```

Table 11      Recognized model types

Type	Description	Type	Description
c	Capacitor model	npn	npn BJT model
csw	Current-controlled switch	pjf	p-channel JFET model
d	Diode model	pmf	p-channel MESFET
l	Mutual inductor model	pmos	p-channel MOSFET model
njf	N-channel JFET model	pnp	pnp BJT model
nmf	N-channel MESFET	r	Resistor model
nmos	N-channel MOSFET model	sw	Voltage-controlled switch

## Chapter 3: Mixed-Mode Simulations

PrimeSim HSPICE Netlist Files

### Examples

```
.MODEL mod1 NPN BF=50 IS=1e-13 VFB=50 PJ=3 N=1.05
```

```
.MODEL mod2 PMOS LEVEL=72
+ aigbinv = 0.0111
+ at = -0.00156
```

### Elements

Element names must begin with a specific letter for each element type.

Table 12     *PrimeSim HSPICE element types that are supported*

First letter	Element	Example
c	Capacitor	Cbypass 1 0 10pf
d	Diode	D7 3 9D1
e	Voltage-controlled voltage source	Ea 1 2 3 4 K
f	Current-controlled current source	Fsub n1 n2 vin 2.0
g	Voltage-controlled current source	G12 4 0 3 0 10
h	Current-controlled voltage source	H3 4 5 Vout 2.0
i	Current source	IA 2 6 1e-6
j	JFET or MESFET	J1 7 2 3 model_jfet w=10u l=10u
k	Linear mutual inductor	K1 L1 L2 0.98
l	Linear inductor	Lx a b 1e-9
m	MOS transistor	M834 1 2 3 4 N1
q	Bipolar transistor	Q5 3 6 7 8 pnp1
r	Resistor	R10 21 10 1000
v	Voltage source	V1 8 0 5
x	Subcircuit call	X1 2 4 17 31 MULTI WN=100 LN=5

## Chapter 3: Mixed-Mode Simulations

### PrimeSim HSPICE Netlist Files

Table 13     *Berkeley SPICE element types that are supported*

First letter	Element	Example
s	Voltage-controlled switch	S1 1 2 3 4 SWITCH1 ON
w	Current-controlled switch	W1 1 2 VCLOCK SWITCHMOD1
z	GaAs MESFET	Z1 7 2 3 ZM1 AREA=2

## Netlist Commands

A limited set of netlist commands is recognized.

To make node names global across all subcircuits, use a .GLOBAL statement. For example:

```
.GLOBAL node1 node2 node3 ...
```

Use the .OPTION PARHIER statement to specify scoping rules. For example:

```
.OPTION PARHIER=GLOBAL | LOCAL
```

Other PrimeSim HSPICE netlist commands that have not been already mentioned explicitly are ignored.

---

## Specifying Physical Devices in Netlists

Sentaurus Device supports an extension of the PrimeSim HSPICE netlist format so that you can specify physical devices in a PrimeSim HSPICE netlist. The .SDEVICE statement declares the name of a physical device and its contacts:

```
.SDEVICE device_name drain gate source bulk
```

Instances of a physical device can be inserted into the netlist using the subcircuit command:

```
x1 d g s b device_name
```

The .SDEVICE statement and a subcircuit instantiation are equivalent to the following specification in the System section of a Sentaurus Device command file:

```
System { device_name x1 (drain=d gate=g source=s bulk=b) }
```

### Note:

PrimeSim HSPICE names are case insensitive, and the netlist parser converts all identifiers to lowercase. Consequently, you must specify physical devices using lowercase in Sentaurus Device.

---

## SPICE Circuit Files

You can specify compact models in an external SPICE circuit file (extension .scf). The declaration of a parameter set can be:

```
PSET pset
  DEVICE dev
  PARAMETERS
    parameter0 = value0
    parameter1 = value1
    ...
  END PSET
```

This declaration introduces the parameter set `pset` that is derived from the device `dev`. It assigns default values for the given parameters. The device `dev` should have already declared the parameter names. Furthermore, the values assigned to the parameters must be of the appropriate type.

*Table 14 Parameters types in SPICE circuit files*

Parameter type	Example	Parameter type	Example
char	c = 'n'	char[]	cc = ['a' 'b' 'c']
double	d = 3.14	double[]	dd = [1.2 -3.4 5e6]
int	i = 7	int[]	ii = [1 2 3]
string	s = "hello world"	string[]	ss = ["hello" "world"]

Similarly, circuit instances can be declared as:

```
INSTANCE inst
  PSET pset
  ELECTRODES e0 e1 ...
  THERMOPLES t0 t1 ...
  PARAMETERS
    parameter0 = value0
    parameter1 = value1
    ...
  END INSTANCE
```

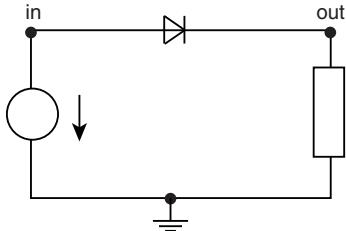
According to this declaration, the instance `inst` is derived from the parameter set `pset`. Its electrodes are connected to the circuit nodes `e0, e1, ...`, and its thermodes are connected to `t0, t1`, and so on. This instance also defines or overrides parameter values.

See *Compact Models User Guide*, Syntax of Compact Circuit (.ccf Files), for the complete syntax of the input language for SPICE circuit files.

---

## Example

Consider the following simple rectifier circuit:



The circuit comprises three compact models and can be defined in the file `rectifier.scf` as:

```
PSET D1n4148
  DEVICE Diode
  PARAMETERS
    is = 0.1p // saturation current
    rs = 16 // Ohmic resistance
    cjo = 2p // junction capacitance
    tt = 12n // transit time
    bv = 100 // reverse breakdown voltage
    ibv = 0.1p // current at reverse breakdown voltage
  END PSET

  INSTANCE v
    PSET Vsource_pset
    ELECTRODES in 0
    PARAMETERS sine = [0 5 1meg 0 0]
  END INSTANCE

  INSTANCE d1
    PSET D1n4148
    ELECTRODES in out
    PARAMETERS temp = 30
  END INSTANCE

  INSTANCE r
    PSET Resistor_pset
    ELECTRODES out 0
    PARAMETERS resistance = 1000
  END INSTANCE
```

The parameter set `D1n4148` defines the parameters shared by all diodes of type `1n4148`. Instance parameters are usually different for each diode, for example, their operating temperature.

**Note:**

You must declare a parameter set before an instance can reference it.

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

The *Compact Models User Guide* further explains the SPICE parameters in this example. The command file for this simulation can be:

```
File { SPICEPath = ". lib spice/lib" }

System { Plot "rectifier" (time() v(in) v(out) i(r 0)) }

Solve {
    Circuit
    NewCurrentPrefix = "transient_"
    Transient (InitialTime = 0 FinalTime = 0.2e-5
                InitialStep = 1e-7 MaxStep = 1e-7) {Circuit}
}
```

The `SPICEPath` in the `File` section is assigned a list of directories. All directories are scanned for SPICE circuit files (`.scf`). Check the log file of Sentaurus Device to see which circuit files were found and used in the simulation.

The `System` section contains a `Plot` statement that produces the `rectifier_des.plt` file and plots the simulation time, the voltages of nodes `in` and `out`, and the current from resistor `r` into node 0.

The `Solve` section describes the simulation. The keyword `Circuit` denotes the circuit equations to be solved.

Instances in a circuit also can appear directly in the `System` section of the command file. For example:

```
System {
    Vsource_pset v (in 0) {sine = (0 5 1meg 0 0)}
    Dln4148 d1 (in out) {temp = 30}
    Resistor_pset r (out 0) {resistance = 1000}

    Plot "rectifier" (time() v(in) v(out) i(r 0))
}
```

---

## Command File for Mixed-Mode Simulations

This section discusses the most important sections of the Sentaurus Device command file for mixed-mode simulations.

---

### System Section

The `System` section defines the netlist of physical devices and circuit elements to be solved. The netlist is connected through circuit nodes.

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

By default, a circuit node is electrical, but it can be declared to be electrical or thermal:

```
Electrical ( enode0 enode1 ... )  
Thermal ( tnode0 tnode1 ... )
```

Each electrical node is associated with a voltage variable, and each thermal node is associated with a temperature variable. Node names are numeric or alphanumeric. Node 0 is predefined as the ground node (0 V).

You can define compact models in PrimeSim HSPICE netlist files (see [PrimeSim HSPICE Netlist Files on page 95](#)) or in SPICE circuit files (see [SPICE Circuit Files on page 101](#)). However, instances of compact models can also appear directly in the `System` section of the command file:

```
parameter-set-name instance-name (node0 node1 ...) {<attributes>}
```

The order of nodes in the connectivity list corresponds to the electrodes and thermodes in the SPICE device definition (see [Compact Models User Guide](#)).

The connectivity list is a list of *contact name=node name* connections, separated by space. The contact name is the name of the contact from the grid file of the given device, and the node name is the name of the circuit netlist node as previously defined in the definition of a circuit element.

Physical devices are defined as:

```
device-type instance-name (connectivity list) {<attributes>}
```

The connectivity list of a physical device explicitly establishes the connection between a contact and a node. For example, the following defines a physical diode and a circuit diode:

```
Diode_pset circuit_diode (1 2) {...} # circuit diode  
Diode243 device_diode (anode=1 cathode=2) {...} # physical diode
```

Both diodes have their anode connected to node 1 and their cathode connected to node 2.

In the case of the circuit diode, the device specification defines the order of electrodes (see [Compact Models User Guide](#), Syntax of Compact Circuit (.ccf) Files). Conversely, the connectivity for the physical diode must be given explicitly. The names `anode` and `cathode` of the contacts are defined in the grid file of the device. The device types of the physical devices must be defined elsewhere in the command file (with `Device` sections) or an external `.device` file.

The `System` section can contain the initial conditions of nodes (see [Initializing Nodes on page 108](#)). Different types of initialization can be specified:

- Fixed values (`Set`)
- Fixed until transient (`Initialize`)
- Initialized only for the first solve (`Hint`)

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

The `System` section can also contain `Plot` statements to generate plot files of node values, currents through devices, and parameters of internal circuit elements. For a simple case of one device without circuit connections (besides resistive contacts), the keyword `System` is not required because the system is implicit and trivial given the information from the `Electrode`, `Thermode`, and `File` sections at the base level.

## Physical Devices

A physical device is instantiated using a previously defined device type, name, connectivity list, and optional parameters. For example:

```
device_type instance-name (<connectivity_list>) {  
    <optional_device_parameters>  
}
```

If no extra device parameters are required, the device is specified without braces. For example:

```
device-type instance-name (<connectivity_list>)
```

The device parameters overload the parameters defined in the device type. As for a device type of Sentaurus Device, the parameters can include `Electrode`, `Thermode`, `File`, `Plot`, `Physics`, and `Math` sections.

### Note:

Electrodes have `Voltage` statements that set the voltage of each electrode. If electrodes are connected to nodes through the connectivity list, these values are only hints (as defined by the keyword `Hint`) for the first Newton solve, but they do not set the electrodes to those values as with the keyword `Set`. By default, an electrode that is connected to a node is floating.

Electrodes must be connected to electrical nodes and thermodes to thermal nodes. This allows electrodes and thermodes to share the same contact name or number.

## Circuit Devices

You can declare SPICE instances in PrimeSim HSPICE netlist files (see [PrimeSim HSPICE Netlist Files on page 95](#)) or in SPICE circuit files (see [SPICE Circuit Files on page 101](#)). They can also appear directly in the `System` section of the command file. For example:

```
pset inst (e0 e1 ... t0 t1 ...) {  
    parameter0 = value0  
    parameter1 = value1  
    ...  
}
```

This declaration in the command file provides the same information as the equivalent declaration in a PrimeSim HSPICE netlist file or a SPICE circuit file.

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

Array parameters must be specified with parentheses, not braces. For example:

```
dd = (1.2 -3.4 5e6)
ss = ("hello" "world")
```

Certain SPICE models have internal nodes that are accessible through the form `instance_name.internal_node`. For example, a SPICE inductance creates an internal node branch, which represents the current through the instance. Therefore, the expression `v(1.branch)` can be used to gain access to the current through the inductor 1. This is useful for plotting internal data or initializing currents through inductors (see the *Compact Models User Guide* for a list of the internal nodes for each model).

## Electrical and Thermal Netlists

Both electrical and thermal netlists can coexist in the same system. For example:

```
System {
    Thermal (ta tc t300)
    Set (t300 = 300)
    Hint (ta = 300 tc = 300)

    Isource_pset i (a 0) {dc = 0}
    PRES pres ("Anode"=a "Cathode"=0 "Anode"=ta "Cathode"=tc)
    Resistor_pset ra (ta t300) {resistance = 1}
    Resistor_pset rc (tc t300) {resistance = 1}

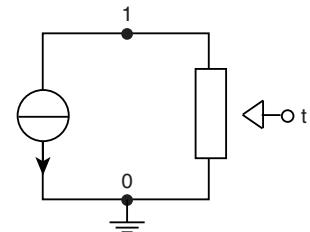
    Plot "pres" (v(a) t(ta) t(tc) h(pres ta) h(pres tc) i(pres 0))
}
```

The current source `i` drives a resistive physical device `pres`. This device has two contacts `Anode` and `Cathode` that are connected to the circuit nodes `a` and `0`, and are also used as thermodes, which are connected to the heat sink `t300` through two thermal resistors `ra` and `rc`.

The `Plot` statement accesses the voltage of node `a`, the temperature of nodes `ta` and `tc`, the heat flux from `pres` into `ta` and `tc`, and the current from `pres` into the ground node `0`.

Many SPICE models provide a temperature parameter for electrothermal simulations. In Sentaurus Device, a temperature parameter is connected to the thermal circuit by a parameter interface.

For example, in this simple circuit, the resistor `r` is a SPICE semiconductor resistor whose resistance depends on the value of the temperature parameter `temp`:



$$r_0 = r_{sh} \cdot \frac{l - narrow}{w - narrow} \quad (12)$$

$$r(temp) = r_0 \cdot (1 + tc_1 \cdot (temp - tnom) + tc_2 \cdot (temp - tnom)^2)$$

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

To feed the value of thermal node  $t$  as a parameter into resistor  $r$ , a parameter interface is required:

```
System {
    Thermal (t)
    Set (t = 300)

    Isource_pset i (1 0) {dc = 1}
    cres_pset r (1 0) {temp = 27 l = 0.01 w = 0.001}
    Param_Interface rt (t) {parameter = "r.temp" offset = -273.15}

    Plot "cres" (t(t) p(r temp) i(r 0) v(1))
}
```

The parameter set `cres_pset` for resistor  $r$  is defined in an external SPICE circuit file:

```
PSET cres_pset
  DEVICE Resistor
  PARAMETERS
    rsh = 1
    narrow = 0
    tc1 = 0.01
    tnom = 27
END PSET
```

The parameter interface `rt` updates the value of `temp` in `r` when the variable  $t$  is changed. This is the general mechanism in Sentaurus Device, which allows a circuit node to connect to a model parameter.

#### Note:

It is important to declare the parameter interface after the instance to which it refers. Otherwise, Sentaurus Device cannot establish the connection between the parameter interface and the instance.

Temperatures in Sentaurus Device are defined in kelvin. SPICE temperatures are measured in degree Celsius. Therefore, an offset of  $-273.15$  must be used to convert kelvin to degree Celsius.

The parameter `Spice_Temperature` in the `Math` section initializes the global SPICE circuit temperature at the beginning of a simulation. It cannot be used to change the SPICE temperature later. To modify the SPICE temperature during a simulation, a *SPICE temperature interface* must be used.

A SPICE temperature interface has one contact that can be connected to an electrode or a thermode. When the value  $u$  of the electrode or thermode changes, the global SPICE temperature is updated according to:

$$\text{Spice temperature} = \text{offset} + c_1 u + c_2 u^2 + c_3 u^3 \quad (13)$$

By default,  $\text{offset} = c_2 = c_3 = 0$  and  $c_1 = 1$ . Therefore, the SPICE temperature interface ensures that the global SPICE temperature is identical to the value  $u$ .

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

In the following example, a SPICE temperature interface is used to ramp the global SPICE temperature from 300 K to 400 K:

```
System {
    Set (st = 300)
    Spice_Temperature_Interface ti (st) { }
}
Solve {
    Quasistationary (Goal {Node=st Value=400} DoZero InitialStep=0.1
        MaxStep=0.1) {
        Coupled {Circuit}
    }
}
```

## Initializing Nodes

The `Set` statement establishes the node value. (The `Unset` statement is used to free a node after a `Set` statement.) This value remains fixed during all subsequent simulations until a `Set` or an `Unset` statement is used in the `Solve` section (see [Mixed-Mode Electrical Boundary Conditions on page 117](#)), or the node becomes a `Goal` of a `Quasistationary`, which controls the node itself (see [Quasistationary in Mixed Mode on page 129](#)). For a set node, the corresponding equation (that is, the current balance equation for electrical nodes and the heat balance equation for thermal nodes) is not solved, unlike an unset node.

### Note:

The `Set` and `Unset` statements exist in the `Solve` section. These act like the `System`-level `Set` but allow more flexibility (see [System Section on page 103](#)).

You can also use the `Set` statement to change the value of a parameter in a compact model. In the following example, the resistance of resistor `r1` changes to 1000 Ω:

```
Set (r1."resistance" = 1000)
```

The `Initialize` statement is similar to the `Set` statement except that node values are kept only during nontransient simulations. When a transient simulation starts, the node is released with its actual value, that is, the node is unset.

### Note:

The `Initialize` statement can be used with an internal node to set a current through an inductor before a transient simulation. For example:

```
Inductor_pset L2 (a b){ inductance = 1e-5 }
Initialize (L2.branch = 1.0e-4)
```

The `Hint` statement provides a guess value for an unset node for the first Newton step only. The numeric value is given in volt, ampere, or kelvin. A current value only makes sense for internal current nodes in circuit devices. The statements are used as follows:

```
Set ( <node> = <float> <node> = <float> ... )
Unset ( <node> <node> ... )
```

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

```
Initialize ( <node> = <float> <node> = <float> ... )
Hint ( <node> = <float> <node> = <float> ... )
```

#### Example

```
Set ( anode = 5 )
```

### Plotting Quantities

The `System` section can contain any number of `Plot` statements to print voltages at nodes, currents through devices, or circuit element parameters. The output is stored in a specified file name. If no file name is provided, the output is sent to the standard output file and the log file. The syntax of the `Plot` statement is:

```
Plot [<filename>] (<plot_command_list>)
```

where `<plot_command_list>` is a list of nodes or plot commands (see [Table 356 on page 1736](#)).

#### Note:

Plot commands are case sensitive.

The `Plot` statement does not print the time by default. When plotting a transient simulation, you must add the `time()` command.

#### Examples

```
Plot (a b i(r1 a) p(r1 rT) p(v0 "sine[0]"))
Plot "plotfile" (time() v(a b) i(d1 a))
```

### Modifying Plot Output

You can use the `ACPlot` statement in the `System` section to modify the output in the Sentaurus Device AC plot file:

```
System {
    ACPlot (<plot_command_list>)
}
```

The `<plot_command_list>` is the same as the plot commands discussed in [Plotting Quantities on page 109](#).

If an `ACPlot` statement is specified, then the given quantities are plotted in the Sentaurus Device AC plot file with the results from the AC analysis. Otherwise, only the voltages at the AC nodes are plotted with the results from the AC analysis.

---

## Device Section

The `Device` sections of the command file define the different device types used in the system to be simulated. Each device type must have an identifier name that follows the keyword `Device`. Each `Device` section can include `Electrode`, `Thermode`, `File`, `Plot`, `Physics`, and `Math` sections. For example:

```
Device "resist" {
    Electrode { ... }
    File { ... }
    Physics { ... }
    ...
}
```

If information is not specified in the `Device` section, then information from the lowest level of the command file is taken (if defined there). For example:

```
# Default Physics section
Physics { ... }
Device resist {
    # This device has no Physics section, so it uses the default set above
    Electrode { ... }
    File { ... }
}
```

---

## File Section

In the `File` section, you can specify the following:

- Name of the output file (keyword `Output`) and name of the small-signal AC extraction file (keyword `ACExtract`)
- Sentaurus Device directory path (keyword `DevicePath`)
- Search path for SPICE models (keyword `SPICEPath`) and compact models (keyword `CMIPath`)
- Default file names for the devices

See [Table 207 on page 1567](#) for the syntax of these keywords.

For the keywords `Output` and `ACExtract`, only a file name without an extension is required. Sentaurus Device automatically appends a predefined extension:

```
File {
    Output = "mct"
    ACExtract = "mct"
}
```

## Chapter 3: Mixed-Mode Simulations

### Command File for Mixed-Mode Simulations

For the keywords `DevicePath`, `SPICEPath`, and `CMIPath`, you must assign a list of directories for which Sentaurus Device searches. For example:

```
File {
    DevicePath = ".../devices:/usr/local/tcad/devices:."
    SPICEPath = ". lib lib/spice"
    CMIPath = ". libcmi"
}
```

Device files (extension `.device`) in `DevicePath` are loaded. The devices found can then be used in the `System` section.

SPICE circuit files (extension `.scf`) in `SPICEPath` are parsed and added to the `System` section of the command file.

Compact circuit files (extension `.ccf`) and the corresponding shared object files (extension `.so.arch`) in `CMIPath` are parsed and added to the `System` section of the command file.

Device-specific keywords are defined under the file entry in the `Device` section.

---

## Math Section

You can specify the following keywords in the global `Math` section for mixed-mode simulations:

```
Math {
    Spice_Temperature=<float>
    Spice_gmin=<float>
}
```

The keyword `Spice_Temperature` is the global SPICE circuit temperature (default: 300.15 K). The keyword `Spice_gmin` refers to the minimum conductance in SPICE (default:  $10^{-12} \Omega^{-1}$ ).

To compute the numeric Jacobian for the SPICE behavioral model (E-element and G-element), you need to specify the following keyword in the global `Math` section:

```
Math { BehavSrcNumericDeriv=0.001 }
```

The keyword `BehavSrcNumericDeriv` is the deviation of the variable. Its default is 0, which means no Jacobian is computed.

---

## Working With Mixed-Mode Simulations

In Sentaurus Device, mixed-mode simulations are handled as a direct extension of single-device simulations.

---

### From Single-Device to Multidevice Command Files

The command file of Sentaurus Device accepts both single-device and multidevice problems. Although the two forms of input look different, they fit in the same input syntax pattern. This is possible because the command file has multiple levels of definitions and there is a built-in default mechanism for the `System` section.

The complete input syntax allows for three levels of device definition: global, device, and instance (see [Figure 8](#)). These levels are linked in that the global level is the default for the device level and instance level.

By default, if no `Device` section exists, a single device is created with the content of a global device. If no `System` section exists, one is created with this single device. In this way, single devices are converted to multidevice problems with a single device and no circuit.

**Note:**

The device that is created by default has the name " " (that is, an empty string).

*Figure 8      Three levels of device definition*

The diagram illustrates the three levels of device definition in a command file. It consists of three vertical columns separated by curly braces (}).

- Global:** The first column contains the text: `File {...}`, `Electrode {...}`, and `...`. A brace on the right side of the column is labeled "Global".
- Device:** The second column contains the text: `Device <type> {`, `File {...}`, `Electrode {...}`, and `...`. A brace on the right side of the column is labeled "Device".
- Instance:** The third column contains the text: `System {`, `<type> <name> {`, `File {...}`, `Electrode {...}`, and `...`. A brace on the right side of the column is labeled "Instance".

```
File {...}
Electrode {...}
...
Device <type> {
    File {...}
    Electrode {...}
    ...
}
System {
    <type> <name> {
        File {...}
        Electrode {...}
        ...
    }
}
```

This translation process can be performed manually by creating a `Device` and `System` section with a single entry (see [Figure 9](#)).

## Chapter 3: Mixed-Mode Simulations

### Working With Mixed-Mode Simulations

Figure 9 Translating single-device syntax to mixed-mode syntax

```
File {
    Output = "diode"
}

Electrode {
    {name="anode"    Voltage=0}
    {name="cathode"  Voltage=1}
}

File {
    Grid   = "diode.tdr"
    Output = "diode"
}

Device diode {
    Electrode {
        {name="anode"    Voltage=0}
        {name="cathode"  Voltage=1}
    }
    File {
        Grid = "diode.tdr"
    }
}

System {
    diode diode1 ...
}
```

The `Solve` section can also be converted if you specify `NoAutomaticCircuitContact`. In [Figure 10](#), all occurrences of `Poisson` must be expanded to `Poisson Contact Circuit`.

Figure 10 Translating default solve syntax to a `NoAutomaticCircuitContact` form

```
Math {
    NoAutomaticCircuitContact
}

Solve {
    Coupled {Poisson Contact Circuit}
    Coupled {Poisson Contact Circuit Electron Hole}
}

Solve {
    Poisson
    Coupled {Poisson Electron Hole}
}
```

---

## File-Naming Conventions for Mixed Mode

You can define a `File` section at all levels of a command file. Therefore, a default file name can be potentially included by more than one device.

In the following example, both `r1` and `r2` use the save device parameters set up in the definition of `res`:

```
Device res {
    File { Save="res" ... }
    ...
}
```

## Chapter 3: Mixed-Mode Simulations

### Working With Mixed-Mode Simulations

```
System {
    res r1
    res r2
}
```

Therefore, in principle, `r1` and `r2` have the same default for `Save` (that is, `res`). Since it is impractical to save both device instances under the same name, names of device instances (that is, `r1` and `r2`) are concatenated to the file name to produce the files `res.r1` and `res.r2`.

This process of file name extension applies to the keywords `Current`, `Plot`, and `Save` of the `File` section, and also when the file name is copied from the global default to a device type declaration.

In practice, the following possibilities exist:

- The file name is defined at the instance level, in which case, it is unchanged.
- The file name is defined at the device level, in which case, the instance name is concatenated to the original file name.
- The file name is defined at the global level of the command file, in which case, the device name and instance name are concatenated to the original file name.

*Table 15 File name extensions for the Current, Plot, and Save keywords*

Level	File name format
Instance	<given name>
Device	<given name>.<instance name>
Global	<given name>.<device type>.<instance name>

# 4

## Performing Numeric Experiments

---

*This chapter describes how to perform numeric experiments.*

Performing a simulation is the virtual analog to performing an experiment in the real world. This chapter describes how to specify the parameters that constitute such an *experiment*, in particular, bias conditions and their variation. It also describes how to perform important types of experiment that involve periodic signals, such as small-signal analysis. Some experiments described here have no real-world analogy, for example, continuous change of physical parameters and device-internal quantities.

---

### Specifying Electrical Boundary Conditions

Electrical boundary conditions are specified in the `Electrode` section. Only one `Electrode` section must be defined for each device. Each electrode is defined in a subsection enclosed by braces and must include a name and default voltage. For example, a complete `Electrode` section is:

```
Electrode {  
    { name = "source" Voltage = 1.0 Current = 1e-3}  
    { name = "drain" Voltage = 0.0 }  
    { name = "gate"   Voltage = 0.0 Material = "PolySi"(P=6.0e19) }  
}
```

[Table 206 on page 1565](#) lists all keywords that can be specified for each electrode. Keywords that relate to the physical properties of electrodes are discussed in [Electrical Boundary Conditions on page 258](#).

Contacts in the `Electrode` section can be specified alternatively by a user-defined regular expression instead of a well-defined name. In this case, all the matching contacts in the structure file (TDR file) will be given the properties defined by the contact with the matching pattern. For example, in a device with nine drain contacts named `drain1` ... `drain9`, you can define them simultaneously with:

```
Electrode {  
    ..  
    { name = regexp("drain[1-9]") Voltage = 0.0 }
```

## Chapter 4: Performing Numeric Experiments

### Specifying Electrical Boundary Conditions

```
    } ...
```

The regular expressions must be specified following the rules defined by the `boost::regex` library with Perl regular expression syntax enabled [1].

Internally, Sentaurus Device matches the regular expressions defined in the `Electrode` section against the contacts in the structure file as a preprocessing step before the simulation. The list of matched structure contacts is expanded as valid contacts, and the simulation proceeds as usual. When structure contacts are matched by more than a regular expression, the last defined in the `Electrode` section succeeds. When a contact is specified in the standard way using the `Name` keyword and also is matched by a regular expression, the last defined in the `Electrode` section succeeds.

You can specify time-dependent boundary conditions, by providing a list of voltage–time pairs. The following example specifies the voltage at `source` to be 5 V at 0 s, increasing linearly to 10 V at 1  $\mu$ s:

```
{ name = "source" voltage = (5 at 0, 10 at 1e-6) }
```

In addition, you can specify a separate *static* voltage. For example:

```
{ name = "source" voltage = 0 voltage = (5 at 0, 10 at 1e-6) }
```

This combination is useful where an initial quasistationary command ramps `source` from 0 V to 5 V, before `source` increases to 10 V during a subsequent transient analysis.

The parameters `Charge` and `Current` determine the boundary condition type as well as the value for an electrode. By default, electrodes have a voltage boundary condition type. The keyword `Current` changes the boundary to current type, and the keyword `charge` changes it to charge type. Note that for current boundary conditions, you still must specify `Voltage`; this value is used as an initial guess when the simulation begins, but it has no impact on the final results. `Charge` and `Current` support time-dependent specifications in the same manner as explained for `Voltage`.

---

## Changing Boundary Condition Type During Simulation

The `Set` statement in the `Solve` section changes the boundary condition type for an electrode as follows:

- To change the boundary condition of a current contact `<name>` to voltage type, use:  

```
Set(<name> mode voltage)
```
- To change the boundary condition of a voltage contact `<name>` to current type or charge type, use one of the following:  

```
Set(<name> mode current)
Set(<name> mode charge)
```

## Chapter 4: Performing Numeric Experiments

### Specifying Electrical Boundary Conditions

- To change the boundary condition of a charge contact <name> to voltage type, use:

```
Set(<name> mode voltage)
```

#### Note:

Changing the boundary condition of a current contact <name> directly to charge type or from a charge contact <name> directly to current type is not allowed.

The following example changes the boundary condition for the voltage contact `drain` from voltage type to current type:

```
Solve { ...
  Set ("drain" mode current)
  ...
}
```

If the boundary condition for `drain` was of current type before the `Set` statement is executed, nothing happens.

The `Set` statement does not change the value of the voltages and currents at the electrodes. The boundary value for the new boundary condition type results from the solution previously obtained for the bias point at which the `Set` statement appears.

Alternatively, to change the boundary condition type of a contact, use the `Quasistationary` statement (see [Quasistationary Ramps on page 122](#)) or the `Transient` statement (see [Transient Ramps on page 144](#)). This is usually more convenient. However, a goal value for the boundary condition must be specified; whereas, the `Set` statement allows you to fix the current, charge, or voltage at a contact to a value reached during the simulation, even if this value is not known beforehand.

---

## Mixed-Mode Electrical Boundary Conditions

In mixed-mode simulations, an additional possibility to specify electrical boundary conditions is to connect electrodes to a circuit (see [Electrical and Thermal Netlists on page 106](#)).

The `Set` statement in the `Solve` section can be used to determine the boundary conditions at circuit nodes (see [Initializing Nodes on page 108](#)). The `Set` statement takes a list of nodes with optional values as parameters. If a value is given, the node is set to that value. If no value is given, the node is set to its current value. The nodes are set until the end of the Sentaurus Device run or the next `Unset` statement with the specified node.

The `Unset` statement takes a list of nodes and *frees* them (that is, the nodes are floating). In practice, the `Set` statement is used in the `Solve` section to establish a complex system of steps, circuit region by circuit region.

The SPICE voltage and current sources use vector parameters to define the properties of various source types. The following example defines a voltage source with an offset

## Chapter 4: Performing Numeric Experiments

### Specifying Thermal Boundary Conditions

`vo = sine[0] = 0.5 V, an amplitude va = sine[1] = 1 V, a frequency freq = sine[2] = 10 Hz, a delay td = sine[3] = 0 s, a damping factor theta = sine[4] = 0 s-1, and a phase shift phi = sine[5] = 0 rad:`

```
System {
    Vsource_pset v0 (n1 n0) { sine = (0.5 1 10 0 0 0) }
}
```

Components of such vectors can be set in the `Solve` section.

The following example sets the offset of the sine voltage source `v0` to 0.75 V:

```
Solve {
    Set (v0."sine[0]" = 0.75) ...
}
```

---

## Specifying Thermal Boundary Conditions

The `Thermode` section defines the thermal contacts of a device. The `Thermode` section is defined in the same way as the `Electrode` section. By default, the temperature specified with `Temperature` is the temperature of the thermode. If, in addition, `Power` (in W/cm<sup>2</sup>) is specified in the `Thermode` section, a heat flux boundary condition is imposed instead, and the specified temperature serves as an initial guess only. For example:

```
Thermode {
    { Name = "top"      Temperature = 350 }
    { Name = "bottom"   Temperature = 300 Power = 1e6 }
}
```

Thermal contacts in the `Electrode` section can be specified alternatively by a user-defined regular expression instead of a well-defined name. In this case, all the matching thermal contacts in the structure file (TDR file) will be given the properties defined by the thermal contact with the matching pattern. The matching process is similar to that of electrical contacts (see [Specifying Electrical Boundary Conditions on page 115](#)).

Time-dependent thermal boundary conditions are specified using the same syntax as for electrical boundary conditions (see [Specifying Electrical Boundary Conditions on page 115](#)).

[Table 358 on page 1738](#) lists the keywords available for the `Thermode` section. [Thermal Boundary Conditions on page 283](#) discusses the physical options. In mixed-mode device simulation, an additional possibility to specify thermal boundary conditions is to connect thermodes to a thermal circuit (see [Electrical and Thermal Netlists on page 106](#)).

### Note:

Only thermodes with a thermal resistive boundary condition can be connected to a circuit. Thermodes with a heat flux boundary condition are not available for circuit connections.

## Chapter 4: Performing Numeric Experiments

### Break Criteria: Conditionally Stopping the Simulation

Table 16     *Various thermode declarations*

Command statement	Description
{ Name = "surface" Temperature=310 SurfaceResistance = 0.1 }	This is a thermal resistive boundary condition with 0.1 cm <sup>2</sup> K/W thermal resistance, which is specified at the thermode 'surface.'
{ Name = "body" Temperature=300 Power = 1e6 }	Heat flux boundary condition.
{ Name = "bulk" Temperature=300 Power = 1e5 Power = (1e5 at 0, 1e6 at 1e-4, 1e3 at 2e-4) }	Thermode with time-dependent heat flux boundary condition.

---

## Break Criteria: Conditionally Stopping the Simulation

Sentaurus Device prematurely terminates a simulation if certain values exceed a given limit. This feature is useful during a nonisothermal simulation to stop the calculations when the silicon starts to melt or to stop a breakdown simulation when the current through a contact exceeds a predefined value.

The following values can be monitored during a simulation:

- Contact voltage (inner voltage)
- Contact current
- Lattice temperature
- Current density
- Electric field (absolute value of field)
- Device power

It is possible to specify values to a lower bound and an upper bound. Similarly, a bound can be specified on the absolute value. The break criteria can have a global or sweep specification. If the break is in sweep, you can have multiple break criteria in one simulation. The break can be specified in a single device and in mixed mode.

## Chapter 4: Performing Numeric Experiments

Break Criteria: Conditionally Stopping the Simulation

---

### Global Contact Break Criteria

The limits for contact voltages and contact currents can be specified in the global `Math` section:

```
Math {
    BreakCriteria {
        Voltage (Contact = "drain" absval = 10)
        Current (Contact = "source" minval = -0.001 maxval = 0.002)
    }
    ...
}
```

In this example, the stopping criterion is met if the absolute value of the inner voltage at the drain exceeds 10 V. In addition, Sentaurus Device terminates the simulation if the source current is less than  $-0.001 \text{ A}/\mu\text{m}$  or greater than  $0.002 \text{ A}/\mu\text{m}$ .

**Note:**

The unit depends on the device dimension:  $\text{A}/\mu\text{m}^2$  for 1D devices,  $\text{A}/\mu\text{m}$  for 2D devices, and  $\text{A}$  for 3D devices.

---

### Global Device Break Criteria

The device power is equal to  $P = \sum_k I_k \cdot V_k$ , where:

- $k$  is the index of a device contact.
- $I_k$  is the current.
- $V_k$  is the inner or outer voltage at this contact.

Examples of the device power criteria are:

```
Math {
    BreakCriteria { DevicePower( Absval=6e-5 ) }
    BreakCriteria { OuterDevicePower( Absval=6e-5 ) }
    BreakCriteria { InnerDevicePower( Absval=2e-6 ) }
    ...
}
```

The keywords `DevicePower` and `OuterDevicePower` are synonyms. In this example, the stopping criterion is met if the absolute value of the outer power exceeds  $6 \times 10^{-5} \text{ W}/\mu\text{m}$  or the inner power exceeds  $2 \times 10^{-6} \text{ W}/\mu\text{m}$ .

## Chapter 4: Performing Numeric Experiments

### Break Criteria: Conditionally Stopping the Simulation

The break criteria for lattice temperature, current density, and electric field can be specified by region and material. If no region or material is given, the stopping criteria apply to all regions.

A sample specification is:

```
Math (material = "Silicon") {
    BreakCriteria {
        LatticeTemperature (maxval = 1000)
        CurrentDensity (maxval = 1e7)
    }
    ...
}
Math (region = "Region.1") {
    BreakCriteria {
        ElectricField (maxval = 1e6)
    }
    ...
}
```

Sentaurus Device terminates the simulation if the lattice temperature in silicon exceeds 1000 K, the current density in silicon exceeds  $10^7$  A/cm<sup>2</sup> or the electrical field in the region Region.1 exceeds  $10^6$  V/cm.

An upper bound for the lattice temperature can also be specified in the `Physics` section. For example:

```
Physics {
    LatticeTemperatureLimit = 1693 # melting point of Si
    ...
}
```

#### Note:

The break criteria of the lattice temperature are only valid for nonisothermal simulations, that is, the keyword `Temperature` must appear in the corresponding `Solve` section.

---

## Sweep-Specific Break Criteria

Sweep-specific break criteria can be specified as an option of `Quasistationary`, `Transient`, and `Continuation`:

```
solve {
    Quasistationary(
        BreakCriteria {
            Current(Contact = "drain" AbsVal = 1e-8)
            DevicePower(Absval = 1e-5)
        }
        Goal { Name="drain" Voltage = 5 }
    )
}
```

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

```
{ coupled { poisson electron hole } }

Quasistationary(
    BreakCriteria { CurrentDensity( AbsVal = 0.1 ) }
    Goal { Name = "gate" Voltage = 2 }
)
{ coupled { poisson electron hole } }
...
}
```

This example contains multiple break criteria. As soon as either the drain current or the device power exceeds the limit, Sentaurus Device stops the first Quasistationary computations and switches to the second Quasistationary. As soon as the current density exceeds its limit, Sentaurus Device exits this section and switches to the next section.

---

## Mixed-Mode Break Criteria

All the previously mentioned break criteria are also available in mixed mode. In this case, the `BreakCriteria` section must contain the circuit device name (an exception are voltage criteria on circuit nodes). Examples of the break criteria conditions in mixed mode are:

```
Quasistationary(
    BreakCriteria {
        # mixed-mode variables
        Voltage( Node = a MaxVal = 10 )
        Current( DevName = resistor Node = b MinVal = -1e-5 )

        # device variables
        Voltage(DevName = diode Contact = "anode" MaxVal=10)
        LatticeTemperature(DevName = mos MaxVal=1000)
        ElectricField(DevName = mos MaxVal=1e6)
        DevicePower(DevName = resistor AbsVal=1e-5)
    }
    ...
)
```

---

## Quasistationary Ramps

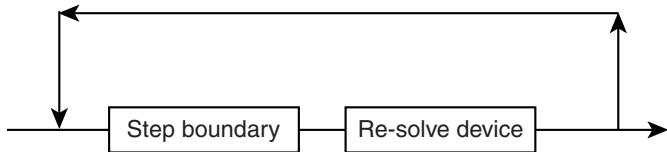
The `Quasistationary` command ramps a device from one solution to another through the modification of its boundary conditions (such as contact voltages) or parameter values.

The simulation continues by iterating between the modification of the boundary conditions or parameter values, and re-solving the device (see [Figure 11](#)). The command to re-solve the device at each iteration is given with the `Quasistationary` command.

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

Figure 11 Quasistationary ramp



---

## Ramping Boundary Conditions

To ramp boundary conditions, such as voltages on electrodes, the Quasistationary command is:

```
Quasistationary( <parameter-list> ) { <solve-command> }
```

The possibilities for `<parameter-list>` are summarized in [Table 348 on page 1729](#).

`<solve-command>` is Coupled, Plugin, ACCoupled, HBCoupled, or possibly another Quasistationary.

For example, ramping a drain voltage of a device to 5 V is performed by:

```
Quasistationary( Goal {Voltage=5 Name=Drain} ){  
    Coupled { Poisson Electron Hole }  
}
```

Internally, the Quasistationary command works by ramping a variable  $t$  from 0.0 to 1.0. The voltage at the contact changes according to the formula  $V = V_0 + t(V_1 - V_0)$ , where  $V_0$  is the initial voltage and  $V_1$  is the final voltage, which is specified in the `Goal` statement.

Control of the stepping is expressed in terms of the  $t$  variable. The control is not made over contact values because more than one contact can be ramped simultaneously.

The step control parameters are `MaxStep`, `MinStep`, `InitialStep`, `Increment`, and `Decrement` in `<parameter-list>`:

- `MaxStep` and `MinStep` limit the step size.
- `InitialStep` controls the size of the first step of the ramping. The step size is automatically augmented or reduced depending on the rate of success of the inner solve command.
- The rate of increase is controlled by the number of performed Newton iterations and by the factor `Increment`.
- The step size is reduced by the factor `Decrement`, when the inner solve fails. The ramping process terminates when the step becomes smaller than `MinStep`.
- If the dynamic nonlocal path band-to-band model is activated, the `NonlocalPath` section can be included (see [Handling Derivatives on page 535](#)).

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

Each contact has a type, which can be voltage, current, or charge. Each Quasistationary command has a goal, which can also be voltage, current, or charge. If the goal and contact type do not match, Sentaurus Device changes the contact type to match the goal. However, Sentaurus Device cannot change a contact of charge type to current type, or a contact of current type to charge type.

Contacts in the Goal section of each Quasistationary can be specified alternatively by a user-defined regular expression instead of a well-defined name. In this case, all the matching valid contacts in the Electrode section will be ramped.

For example, the following statement will ramp all contacts named d1 ... d9 to 5 V:

```
Quasistationary( Goal {Voltage=5 Name=Regexp("d[1-9]") } ){  
    Coupled { Poisson Electron Hole }  
}
```

For details about the matching process, see [Specifying Electrical Boundary Conditions on page 115](#).

The initial value (for  $t = 0$ ) is the current or voltage computed for the contact before the Quasistationary command starts. Contacts keep their boundary condition type after the Quasistationary command finishes. To change the boundary condition type of a contact explicitly, use the Set command (see [Changing Boundary Condition Type During Simulation on page 116](#)).

If all equations to be solved in a Quasistationary also have been solved in the solve statement immediately before, Sentaurus Device omits the point  $t = 0$ , as the solution is already known. Otherwise, this point is computed at the beginning of the Quasistationary. You can explicitly force or prevent this point from being computed using DoZero or -DoZero in the <parameter-list>.

---

## Ramping Quasi-Fermi Potentials in Doping Wells

Electron and hole quasi-Fermi potentials in specified doping wells can also be ramped in a Quasistationary statement. This is a quick and robust way to deplete doping wells without having to solve the continuity equation for the carrier to be depleted. Its main application is in CMOS image sensor simulations and charge-coupled device simulations.

To ramp a quasi-Fermi potential, specify the keyword WellContactName or DopingWell followed by eQuasiFermi or hQuasiFermi in the Goal section of the Quasistationary command:

```
Goal {  
    [WellContactName= <contact_name> | DopingWell(<point_coordinates>)]  
    [eQuasiFermi= <value> | hQuasiFermi= <value>]  
}
```

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

The `DopingWell` specification is more general and can be used for both semiconductor wells with or without contacts (buried wells). In this case, the coordinates of a point in the well must be given to select the well (see previous code).

When the semiconductor well where the quasi-Fermi potential is ramped is connected to a contact, the `WellContactName` specification can be used. Now, the contact associated with the well must be specified to select the well (see previous code).

The starting value of the `Goal` for the quasi-Fermi potential ramping in the specified well is taken as the well averaged value of the quasi-Fermi potential at the end of the previous `Quasistationary` command.

This starting value can be changed using a `Set` statement:

```
Solve {
    Set(DopingWell(0.5 0.5) eQuasiFermi=5.0)
    Quasistationary(... Goal {DopingWell(0.5 0.5) eQuasiFermi=10.0})
        ) {Coupled {poisson hole}}
}
```

The `Set` statement has the same options and syntax as the `Goal` statement above.

In mixed-mode simulations, the feature still can be used for each device separately. In this case, the keywords `DopingWell` and `WellContactName` must be prefixed by the device name:

```
System {
    MOSCAP1 "mc1" (
        "gate_contact" = gate1
        "substrate_contact" = sub
    )
    ...
}
Solve {
    Set(mc1.DopingWell(0.5 0.5) eQuasiFermi=5.0)
    Quasistationary...
        Goal {mc1.DopingWell(0.5 0.5) eQuasiFermi=10.0}
            ) {Coupled {poisson hole circuit}}
}
```

In addition, Sentaurus Device supports multiple-well ramping, activated by the keyword `DopingWells` with an option in parentheses. The syntax is:

```
Quasistationary...
    Goal {
        DopingWells([Region=<string> | Material=<string> | Semiconductor])
        eQuasiFermi=<value>
    }
) {Coupled {poisson hole}}
```

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

As can be seen from this syntax description, wells in a region, wells having the same material, or all semiconductor wells in the device can be ramped to a specified value of the electron or hole quasi-Fermi potential. A well is considered to be in a region if it has one or more vertices in common with the region. A well belongs to a region even if it is not entirely in the region.

For multiple-well ramping, the `Set` command is used to change the quasi-Fermi potential in a group of wells before a Quasistationary ramping:

```
Set(DopingWells([Region=<string> | Material=<string> | Semiconductor])
eQuasiFermi=5.0)
```

In the case of mixed-mode simulations, `DopingWells` must be prefixed with the device instance name and a dot (.) similar to single-well syntax.

You can set the quasi-Fermi potential in a user-defined window as follows:

```
Set (DopingWells [(x y z) (x1 y1 z1) (x2 y2 z2)] eQuasiFermi=5.0)
```

The vector `(x y z)` is used to locate the doping well to be modified. The other two vectors `(x1 y1 z1)` and `(x2 y2 z2)` are the corners of the user-defined window. Only the overlapping area between the selected doping region and the user-defined window will have the quasi-Fermi value defined in the command file.

Furthermore, instead of using the vector `(x y z)` to locate the doping well, you can use the region name, material name, or material group name to locate the doping well. For example:

```
Set (DopingWells [<region> | <material> | <Semiconductor> (x1 y1 z1)
(x2 y2 z2)] eQuasiFermi=5.0)
```

With this syntax, the overlapping area between the selected region, material, or semiconductor and the user-defined window will have the quasi-Fermi value defined in the command file.

Electron and hole quasi-Fermi potentials can be simultaneously ramped using multiple `Goal` statements and solving for the Poisson equation only.

When the continuity equation has been solved for a carrier whose quasi-Fermi potential is to be ramped, the initial value of the quasi-Fermi potential for all vertices in the well is computed from the carrier density in the point specified when the well was defined. For multiple-well ramping, the same scheme applies well-wise.

---

## Ramping Physical Parameter Values

A Quasistationary command allows parameters from the parameter file of Sentaurus Device to be ramped.

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

The `Goal` statement has the form:

```
Goal {  
  [ Device=<device> ]  
  [ Material=<material> | MaterialInterface=<interface> |  
    Region=<region> | RegionInterface=<interface> ]  
  Model=<model> Parameter=<parameter> Value=<value>  
}
```

Specifying the device and location (material, material interface, region, or region interface) is optional. However, the model name and parameter name must always be specified.

A list of model names and parameter names is obtained by using:

```
sdevice --parameter-names
```

This list of parameters corresponds to those in the Sentaurus Device parameter file, which can be obtained by using `sdevice -P` (see [Generating a Copy of Parameter File on page 80](#)).

The following command produces a list of model names and parameter names that can be ramped in the command file:

```
sdevice --parameter-names <command_file>
```

Sentaurus Device reads the devices in the command file and reports all parameter names that can be ramped. However, no simulation is performed. The models in [Table 17](#) contain command file parameters that can be ramped.

*Table 17 Command file parameters*

Model name	Parameters
DeviceTemperature	Temperature
GalvanicTransport	MagneticFieldx, MagneticFieldy, MagneticFieldz
Optics	Wavelength, Intensity, Theta, Phi, Polarization
PEPolarization	activation
RadiationBeam	Dose, DoseRate, DoseTSigma, DoseTime_end, DoseTime_start
Strain	StrainXX, StrainXY, StrainXZ, StrainYY, StrainYZ, StrainZZ
Stress	StressXX, StressXY, StressXZ, StressYY, StressYZ, StressZZ
Traps(<index>)	Conc, EnergyMid, EnergySig, eGfactor, eJfactor, eXsection, hGfactor, hJfactor, hXsection

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

#### Note:

Certain models such as traps must be specified with an index. For example:

```
Model = "Traps(1)"
```

The index denotes the exact model for which a parameter should be ramped. Usually, Sentaurus Device assigns an increasing index starting with zero for each optical beam, trap, and so on. However, the situation becomes more complex if material and region specifications are present. To confirm the value of the index, using the following command is recommended:

```
sdevice --parameter-names <command file>
```

Mole fraction-dependent parameters can be ramped. For example, if the parameter  $p$  is mole fraction-dependent, the parameter names listed in [Table 18](#) can appear in a `Goal` statement.

*Table 18 Mole fraction-dependent parameters as Quasistationary Goals*

Parameter in Goal statement	Description
Parameter= $p$	Parameter $p$ in non-mole fraction-dependent materials.
Parameter=" $p(0)$ " Parameter=" $p(1)$ " ...	Interpolation value of $p$ at $Xmax(0)$ , $Xmax(1)$ , ...
Parameter=" $B(p(1))$ " Parameter=" $C(p(1))$ " Parameter=" $B(p(2))$ " Parameter=" $C(p(2))$ " ...	Quadratic and cubic interpolation coefficients of $p$ in intervals $[Xmax(0), Xmax(1)]$ , $[Xmax(1), Xmax(2)]$ , ...

Mole fraction-dependent parameters can be ramped in all materials. In mole fraction-dependent materials, interpolation values  $p(\dots)$  and interpolation coefficients  $B(p(\dots))$  and  $C(p(\dots))$  must be ramped. In a non-mole fraction-dependent material, only the parameter  $p$  can be ramped.

If a parameter is not found, then Sentaurus Device issues a warning, and the corresponding goal statement is ignored.

Parameters in PMI models can also be ramped.

To ramp optical parameters, see [Parameter Ramping on page 689](#).

## Quasistationary in Mixed Mode

The Quasistationary statement is extended in mixed mode to include goals on nodes and circuit model parameters. [Table 343 on page 1724](#) shows the syntax of these goals.

The goal is usually used on a node that has been fixed with the Set or Initialize statement in the System section. The statement Goal{Node=<string> Voltage=<float>} assigns a new target voltage to a specified node. For example, the node a is set to 1 V and ramped to 10 V:

```
System {
    Resistor_pset r1(a 0){resistance = 1}
    Set(a=1)
}
Solve {
    Circuit
    Quasistationary( Goal{Node=a Voltage=10} ){ Circuit }
}
```

A goal on circuit model parameters can be used to change the configuration of a system. The statement Goal{Parameter=<i-name>.<p-name> value=<float>} assigns a new target value to a specified parameter *p-name* of a device instance *i-name*. Any circuit model parameter can be changed.

For example, the resistor r1 is ramped from 1 Ω to 0.1 Ω, and the offset of the sine voltage source is ramped to 1.5 V:

```
System {
    Resistor_pset r1(a 0){resistance = 1}
    Vsource_pset v0 (n1 n0) { sine = (0.5 1 10 0 0 0) }
    Set(a=1)
}
Solve {
    Circuit
    Quasistationary(
        Goal { Parameter = r1."resistance" Value = 0.1 }
        Goal { Parameter = v0."sine[0]"      Value = 1.5 })
    { Circuit }
}
```

**Note:**

When a node is used in a goal, it is set during the quasistationary simulation. At the end of the ramp, the node reverts to its previous set status, that is, if it was not set before, it is not set afterward. This can cause unexpected behavior in the Solve statement following the Quasistationary statement. Therefore, it is better to set the node in the Quasistationary statement using the Set command.

## Saving and Plotting During a Quasistationary

Data can be saved and plotted during a Quasistationary ramping process by using the `Plot` command. `Plot` is placed with the other Quasistationary parameters. For example:

```
Quasistationary(
    Goal {Voltage=5 Name=Drain}
    Plot {Range = (0 1) Intervals=5})
    {Coupled{ Poisson Electron Hole }}
```

In this example, six plot files are saved at five intervals:  $t = 0, 0.2, 0.4, 0.6, 0.8,$  and  $1.0.$

Data can also be saved and plotted when a specific step has passed since last save by using the keyword `MinSpacing` in the `Plot` command. For example:

```
Quasistationary(
    Goal {Voltage=5 Name=Drain}
    Plot {Range = (0 1) MinSpacing=0.2}
)
{Coupled{ Poisson Electron Hole }}
```

In this example, no extra intervals are added into the simulation. Within the range  $(0 1),$  those ramping points that are  $0.2$  larger than the last saved point are plotted and saved. After `MinSpacing` is defined in the command file, other plot commands will be ignored.

Another way to plot data is with `Plot` in the Quasistationary body rather than inside the Quasistationary parameters (see [When to Plot on page 178](#)). `Plot` is added after the equations to solve, such as:

```
Quasistationary( Goal {Voltage=5 Name=Drain} ){
    Coupled { Poisson Electron Hole }
    Plot ( Time= ( 0.2; 0.4; 0.6; 0.8; 1.0 ) NoOverwrite )
}
```

**Note:**

When the distance between the specified save point and the time step is smaller than the numeric accuracy, the interval can be omitted by the `Plot` statement and the file is not written. Use the `SaveWithinMinStep` keyword in the global `Math` section to save all of the specified points.

---

## Extrapolation

The Quasistationary command can use extrapolation to predict the next solution based on the values of the previous solutions. Extrapolation can be switched off (default) or on globally in the `Math` section:

```
Math { -Extrapolate } # default
```

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

or:

```
Math { Extrapolate }
```

Alternatively, you can switch it off or on for a specific Quasistationary command only:

```
Quasistationary (
    Goal { ... }
    -Extrapolate
) { Coupled { Poisson Electron Hole } }
```

Sentaurus Device can use extrapolation in the following cases: inside a Quasistationary command and between Quasistationary commands.

### Inside a Quasistationary Command

The initial guess for a given step is obtained by extrapolation from the solutions of the previous steps (if they exist).

### Between Quasistationary Commands

The extrapolation information is preserved between Quasistationary commands, and it also is saved and loaded automatically by the `Save` and `Load` commands. A subsequent Quasistationary command uses this extrapolation information if the following conditions are met:

1. The previous and current Quasistationary commands have the same number of goals.
2. The previous and current Quasistationary commands ramp the same quantities.
3. The previous and current Quasistationary commands are contiguous, that is, for all goals, the current initial value is equal to the goal value in the previous command.
4. Multiple goals are ramped at the same rate in the current Quasistationary command compared to the previous Quasistationary command. As an example, assume that all contact voltages have zero initial values. Then, the following two Quasistationary commands satisfy this condition:

```
Quasistationary (
    Goal { Name = "gate"    Voltage = 1 }
    Goal { Name = "drain"   Voltage = 2 }
) { Coupled { Poisson Electron Hole } }

Quasistationary (
    Goal { Name = "gate"    Voltage = 3 }
    Goal { Name = "drain"   Voltage = 6 }
) { Coupled { Poisson Electron Hole } }
```

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

On the other hand, the following two Quasistationary commands violate this condition:

```
Quasistationary (
    Goal { Name = "gate"    Voltage = 1 }
    Goal { Name = "drain"   Voltage = 2 }
) { Coupled { Poisson Electron Hole } }

Quasistationary (
    Goal { Name = "gate"    Voltage = 3 }
    Goal { Name = "drain"   Voltage = 10 }
) { Coupled { Poisson Electron Hole } }
```

In the second Quasistationary command, the drain voltage is ramped at a higher rate than the gate voltage. Therefore, the extrapolation information from the previous Quasistationary command cannot be used.

5. The values of the solution variables have not changed between the two Quasistationary commands, for example, by a `Load` command.

If the extrapolation information from a previous Quasistationary command can be used successfully, the following message appears in the log file:

```
Reusing extrapolation from a previous quasistationary
```

The options in [Table 19](#) control the handling of extrapolation information between Quasistationary commands.

*Table 19 Extrapolation options*

Option	Description
<code>ReadExtrapolation</code>	Tries to use the extrapolation information from a previous command if it is available and compatible. This is the default.
<code>-ReadExtrapolation</code>	Do not use the extrapolation information from a previous command.
<code>StoreExtrapolation</code>	Stores the extrapolation information internally at the end of the command, so that it will be available for a subsequent command, or can be written to a save or plot file. This is the default.
<code>-StoreExtrapolation</code>	Do not store the extrapolation information internally at the end of the command.

## Parameters of the Extrapolate Statement

This section discusses the parameters of the `Extrapolate` statement.

### Order Extrapolation

By default, Sentaurus Device uses linear extrapolation, but you can request a higher order extrapolation. For example, to specify quadratic extrapolation, use:

```
Extrapolate (Order = 2)
```

### Exclude Equations

By default, Sentaurus Device uses extrapolation for all equations from a `Coupled` command, but you can exclude some equations from this list.

For example, to extrapolate only for the `Electron` equation with second-order extrapolation, use:

```
Quasistationary (
    Goal { Name = "gate" Voltage = 3 }
    Extrapolate(Order=2 Exclude(Poisson Hole))
) { Coupled { Poisson Electron Hole } }
```

### Low Density Limit

If the electron or hole concentration has small values in some vertices, extrapolation in these vertices might be the reason for convergence problems. You can define a low limit for these concentrations starting from which Sentaurus Device will extrapolate the corresponding density.

For example, to extrapolate the carrier density only in vertices in which the corresponding concentration is more than `1e+08`, use:

```
Quasistationary (
    Goal { Name = "gate" Voltage = 3 }
    Extrapolate(LowDensityLimit = 1e+8)
) { Coupled { Poisson Electron Hole } }
```

### Minimum Step of Extrapolation

The `Quasistationary` or `Transient` commands contain the options `Initialstep`, `Increment`, `Decrement`, `Maxstep`, and `Minstep`. Sentaurus Device uses the following iterative procedure.

#### Procedure 1:

1. An initial guess for the given step ( $t_2=t_1+step$ ) is obtained by extrapolation from the solutions of the previous steps (that is, it is dependent on  $t_0$ ,  $t_1$ , and `step`).
2. Sentaurus Device uses the Newton iterative method to obtain the solution for  $t=t_2$ .

## Chapter 4: Performing Numeric Experiments

### Quasistationary Ramps

3. **IF** this iterative process converges, **THEN**  $t_0=t_1$ ,  $t_1=t_2$ , and  $\text{step}=\text{NewStep}$  (there is a special algorithm for computing `NewStep`). **GOTO 1**.
4. **ELSE**  $\text{step}=\text{step}/\text{Decrement}$ , **IF**  $\text{step} > \text{Minstep}$  (it is a Quasistationary parameter). **GOTO 1**.
5. **ELSE** (that is,  $\text{step} < \text{Minstep}$ ) **Procedure 1** has not converged. In this case, Sentaurus Device exits from the Quasistationary or Transient commands.

Sometimes, for small steps, **Procedure 1** has better convergence without extrapolation. For this case, the `Extrapolate` statement has the `MinStep` parameter, which is the minimum step of the extrapolation (default value is 0).

For example:

```
Quasistationary (
    Minstep = 1e-8                      # it is Quasistationary minimum step
    Extrapolate(MinStep = 1e-5)          # it is Extrapolate minimum step
    Goal { Name = "gate" Voltage = 3 }
) { Coupled { Poisson Electron Hole } }
```

Therefore, item 1 of **Procedure 1** changes to the following:

#### Algorithm=1

- 1a) **IF**  $\text{step} > \text{Extrapolate.MinStep}(1e-5)$ , **THEN** an initial guess for the given step ( $t_2=t_1+\text{step}$ ) is obtained by extrapolation from the solutions of the previous steps.
- 1b) **ELSE IF**  $\text{step} > \text{Quasistationary.Minstep}(1e-8)$ , **THEN** Sentaurus Device uses the current solution (not extrapolated) as the initial guess.
- 1c) **ELSE GOTO 5**.

#### Note:

If `Extrapolate.MinStep` is greater than `Quasistationary.Maxstep`, then Sentaurus Device does not use extrapolation, that is, this is equivalent to using the `Quasistationary` command without the `Extrapolate` statement.

#### Number of Failed Extrapolations

In addition to, or as an alternative to `MinStep`, Sentaurus Device can use the number of successive failures to switch off extrapolation and to use the current solution as the initial guess. For example:

```
Extrapolate( NumberOfFailures = 2 )
```

In this case, after **two consecutive** failures with extrapolation, Sentaurus Device uses the current solution (not extrapolated) as the initial guess.

## Chapter 4: Performing Numeric Experiments

### Continuation Command

#### Switch Algorithm

For example:

```
Extrapolate( MinStep=1e-5 NumberOfFailures=2 Algorithm=<1 | 2> )
```

The default is `Algorithm=1`, which means **Procedure 1** is used. That is, if `Step < MinStep` or the number of consecutive failures  $\geq$  `NumberOfFailures`, then Sentaurus Device uses **only** the current solution (not extrapolated) as the initial guess.

If `Algorithm=2`, then Sentaurus Device uses extrapolation in any case. **Only if it fails** does Sentaurus Device switch to the not extrapolated solution as the initial guess.

---

## Continuation Command

The Continuation command enables automated tracing of arbitrarily shaped I–V curves. This capability is particularly useful for tracing I–V curves associated with complicated device phenomena such as breakdown or latchup, which can be multivalued with abrupt changes.

The implementation is based on a dynamic load-line technique [2] adapting the boundary conditions along the I–V curve to ensure convergence. The boundary condition consists of an external voltage applied to the end of a variable load resistor connected to the device electrode for which the I–V curve is traced. The program calculates an optimal boundary condition at each point along the I–V curve by adjusting the load line so it is orthogonal to the local tangent of the I–V curve.

The continuation method is activated by specifying the keyword `Continuation` in the `Solve` section. Control parameters that define the setup must be given in parentheses that follow the `Continuation` keyword.

For example:

```
Solve {
    Continuation (<Control Parameters>) {
        Coupled { poisson electron hole }
    }
}
```

[Table 339 on page 1718](#) summarizes the control parameters of the `Continuation` command. The method works with both single-device and mixed-mode setups, with only one electrode undergoing continuation at a time.

## Continuation Control Parameters

**Note:**

Specifying the `NewArc` option will invoke several features that improve the robustness of the method for most applications. If you do not specify this option, the original continuation algorithm will be used by default.

The first step of a continuation simulation is always a voltage-controlled step specified with the `InitialVstep` parameter. After this step, the I–V curve tracing proceeds automatically until the current or voltage at the continuation electrode advances outside of a user-defined continuation window specified with the parameters `MinVoltage`, `MaxVoltage`, `MinCurrent`, and `MaxCurrent`. For example:

```
Solve { ...
    Continuation (
        NewArc
        Name="collector" InitialVstep=0.1
        MinVoltage=0  MaxVoltage=10
        MinCurrent=0  MaxCurrent=1e-3
    )
    { Coupled { poisson electron hole } }
}
```

This code block specifies that the I–V curve must be traced at the collector electrode, the initial voltage-controlled step is 0.1 V, the voltage range is 0 V to 10 V, and the current range is 0 A to 1 mA.

The step size along the I–V curve is controlled primarily by the convergence of the simulation. The step size increases by a factor `Increment` (default is 1.5) if a solution converges, and it decreases by a factor `Decrement` (default is 2.0) if it does not converge.

You can limit the step size by specifying the parameters `MaxVstep`, `MaxIstep`, and `MaxIfactor`. These parameters limit the projections  $\Delta V$  and  $\Delta I$  calculated by Sentaurus Device from one point along the I–V curve to the next. The actual maximum spacing between successive points might differ slightly from these limits. The parameters `MaxVstep` and `MaxIstep` represent the maximum voltage and current steps, respectively, while `MaxIfactor` represents the maximum factor that the current is allowed to increase in one step.

During continuation, Sentaurus Device automatically calculates an optimal value for the load resistance that is attached to the continuation electrode. Limits for the calculated resistance can be specified using the `MinRload` and `MaxRload` parameters.

Tracing an I–V curve successfully with the continuation method depends on how accurately the local slope of the traced I–V curve is computed. At low biases, the current at the continuation electrode can be small and noisy. This can result in an inaccurate computation of the slope, which can sometimes cause the continuation method to backtrace. In such cases, it might be necessary to specify the parameter `Vadapt`, or `Iadapt`, or both that

## Chapter 4: Performing Numeric Experiments

### Continuation Command

represent the minimum voltage and the current, respectively, that must be reached before the adaptive algorithm is switched on. From `MinVoltage` to `Vadapt` and from `MinCurrent` to `Iadapt`, the adaptive algorithm is switched off and the simulation proceeds as a simple voltage ramping through a fixed-value resistor attached to the continuation electrode. The default value for the fixed resistor is  $0.001 \Omega$  and can be changed using the `Rfixed` parameter.

---

## Continuation in Mixed Mode

The continuation method can be used in mixed-mode simulations; however, the continuation contact must not be connected to any circuit node. When using continuation in mixed mode, the device instance must be prefixed to the `Name` parameter.

In the following example, consider a device with the identifier `mos1`, with electrodes named `source`, `drain`, `gate`, and `substrate`. The device instance for `mos1` (given in the `System` section) is `d1`. If `drain` should be used as the continuation electrode, no circuit connection should be made to it in the connectivity list, and it must be identified using `d1.Name` in the `Continuation` section:

```
Device mos1 {
    Electrode {
        {Name="source" Voltage=0.0}
        {Name="drain" Voltage=0.0}
        {Name="gate" Voltage=0.0}
        {Name="substrate" Voltage=0.0}
    }
    ...
}
System {
    mos1 d1 (source=s1 gate=g1 substrate=s1)
    set (s1=0 g1=0 b1=0)
}
Solve {
    ...
    Continuation(
        d1.Name="drain"
        ...
    ) {Coupled {Poisson Electron Hole}}
}
```

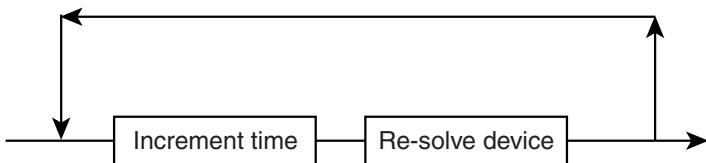
To bias a mixed-mode continuation electrode before `Continuation`, you can use a `Quasistationary` ramp. However, the electrode must be identified using the `Contact` keyword instead of `Name` to avoid biasing the contact as a circuit node. For example:

```
Quasistationary (
    InitialStep=1e-3
    ...
    Goal {Contact=d1."drain" Voltage=3}
) {Coupled {Poisson Electron Hole}}
```

## Transient Command

The Transient command is used to perform a transient time simulation. The command must start with a device that has already been solved. The simulation continues by iterating between incrementing time and re-solving the device (see [Figure 12](#)). The command to solve the device at each iteration is given with the Transient command.

*Figure 12* Transient simulation



The syntax of the Transient command is:

```
Transient ( <parameter-list> ) { <solve-command> }
```

[Table 352 on page 1732](#) lists the possible parameters and <solve-command> is Coupled or Plugin.

An example of performing a transient simulation for 10  $\mu$ s is:

```
Transient( InitialTime = 0.0 FinalTime=1.0e-5 ){
    Coupled { Poisson Electron Hole }
}
```

The Transient command allows you to overwrite time-step control parameters, which have default values or are globally defined in the Math section. The error control parameters that are accepted by the Transient command are listed in [Table 232 on page 1615](#).

The parameters TransientError, TransientErrRef, and TransientDigits control the error over the transient integration method. This differs from the error control for the Coupled statement, which only controls the error of each nonlinear solution. As with the error parameters for the Coupled statement, the transient error controls can be both absolute and relative. Absolute values are parameterized according to equation-variable type.

The plot controls of the Transient command are the same as for the Quasistationary command, except the  $t$  is the real time (in seconds), and is not restricted to an interval from 0 to 1.

Plot can be given as a Transient parameter. For example:

```
Transient( InitialTime = 0.0 FinalTime = 1.0e-5
    Plot { Range = (0 3.0e-6) Intervals=3 } )
    { Coupled { Poisson Electron Hole } }
```

This example saves four plot files at  $t = 0.0, 1.0e-6, 2.0e-6$ , and  $3.0e-6$ .

## Chapter 4: Performing Numeric Experiments

### Transient Command

Similar to the Quasistationary simulation, data can also be saved and plotted when a specific step has passed since the last save by using the keyword `MinSpacing` in the `Plot` command. For example:

```
Transient(
    InitialTime = 0.0 FinalTime = 1.0e-5
    Plot { Range = (0 3.0e-6) MinSpacing= 2.0e-5 }
)
{ Coupled { Poisson Electron Hole } }
```

In this example, no extra intervals are added into the simulation. Within the range (0 3.0e-6), those ramping points that are 2.0e-5 larger than the last saved point are plotted and saved. After `MinSpacing` is defined in the command file, other plot commands will be ignored.

Alternatively, `Plot` can be put into the `Transient` solve command (see [When to Plot on page 178](#)):

```
Transient( InitialTime = 0.0 FinalTime = 1.0e-5 )
{
    Coupled{ Poisson Electron Hole }
    Plot ( Time=( 1.0e-6; 2.0e-6; 3.0e-6 ) NoOverwrite )
}
```

---

## Numeric Control of Transient Analysis

You can control transient simulations by using a set of keywords available in the `Math` section and the options of `Transient` statements in the `Solve` section. Sentaurus Device uses implicit discretization of nonstationary equations and supports two discretization schemes: the trapezoidal rule/backward differentiation formula (TRBDF), which is the default, and the simpler backward Euler (BE) method.

To select a particular transient method, specify `Transient=<string>`, where `<string>` can be `TRBDF` or `BE`.

In transient simulations, in addition to numeric errors of the nonlinear equations, discretization errors due to the finite time-step occur (see [Transient Simulation on page 1199](#)). By default, Sentaurus Device does not control the time step to limit this discretization error (that is, that time step depends only on convergence of Newton iterations).

To activate time-step control, `CheckTransientError` must be specified. For time-step control, Sentaurus Device uses a separate set of criteria, but as with Newton iterations, the control of both relative and absolute errors is performed.

The relative transient error  $\epsilon_{R,tr}$  is defined similarly to  $\epsilon_R$  in [Equation 32 on page 194](#):

$$\epsilon_{R,tr} = 10^{-\text{TransientDigits}} \quad (14)$$

## Chapter 4: Performing Numeric Experiments

### Transient Command

Similarly, for  $x_{\text{ref,tr}}$  and  $\varepsilon_{A,\text{tr}}$ , the following equation is valid:

$$x_{\text{ref,tr}} = \frac{\varepsilon_{A,\text{tr}}}{\varepsilon_{R,\text{tr}}} x^* \quad (15)$$

The keyword `TransientDigits` must be used to specify relative error  $\varepsilon_{R,\text{tr}}$  in transient simulations. An absolute error in time-step control can be specified by either the keyword `TransientError` ( $\varepsilon_{A,\text{tr}}$ ) or `TransientErrRef` ( $x_{\text{ref,tr}}$ ), and their values can be defined independently for each equation variable. The same flag as in Newton iteration control, `RelErrControl`, is used to switch to unscaled (`TransientErrRef`) absolute error specification.

For simulations with floating gates, Sentaurus Device can monitor the errors in the floating-gate charges as well. For details, see [Floating Gates on page 1202](#).

#### Note:

All transient parameters in the `Math` section, except `Transient`, can be overwritten in the `Transient` command of the `Solve` section (see [Transient Command on page 138](#)).

The `Transient` command uses time-dependent boundary conditions (see [Specifying Electrical Boundary Conditions on page 115](#)) and correspondingly updates the quasi-Fermi potentials in doping wells. This might interfere with previously ramped ones in the `Quasistationary` command (see [Ramping Quasi-Fermi Potentials in Doping Wells on page 124](#)). This is critical for a Poisson-only solution. To avoid this, use `-UpdateQFPinTransientPoisson` in the `Math` section.

---

## Time-Stepping

A transient simulation computes the status of the system or device as a function of time for a finite time range, specified by `InitialTime` and `FinalTime`, by advancing from a status at a given time to the status at a later time point. The time step is chosen dynamically. The time step is bounded by `MinStep` and `MaxStep` from below and above, respectively; while `InitialStep` specifies the first time step at start time.

If the computation of the system status for a given time step fails, the time step is reduced iteratively until either the system status can be computed successfully or the minimum time step is reached. In the later case, the `Transient` terminates with an error. The factor by which a step is reduced can be set with the `Decrement` keyword.

After a successful computation of the system status, the time step is typically increased, depending on the computational effort for the actual time step. The maximum factor by which a step is increased can be controlled with the `Increment` keyword.

The effective time step can be smaller than the advancing time step, for example, if time points are specified for plotting, where a system status must be computed. However, these

## Chapter 4: Performing Numeric Experiments

### Transient Command

effective time steps do not reduce the advancing time step, that is, the subsequent effective time step might be as large as the advancing time step. You can bound (from above) the advancing time step at certain conditions by specifying turning points as a parameter in the Transient.

These conditions can be a list of arbitrary time points or time ranges (see [Table 352 on page 1732](#)). For example:

```
Transient ( ...
    TurningPoints (
        (Condition (Time ( 1.e-7 ; 2.e-7 ; 3.e-7 ) ) Value= 1.1e-9 )
        (Condition (Time ( Range= (1.e-7 2.e-7 ) ) ) Value= 2.1e-9 )
        (Condition (Time ( 1.23e-7 ; 1.45e-7 ; 1.67.e-7 ) ) Value= 1.0e-10 )
    )
)
```

The advancing time step is limited by the specified `Value` (in seconds) for the corresponding `Condition`. Here, the first condition limits the value for a list of time points. The second condition limits the time step for a whole range. If within a time range (where you have specified a maximum time step, like the second condition in this example) and you want to output results at specific time points within that range, simply specify these in another `Condition` statement (third statement in the example), ensuring that you set a `Value` that is less than the `Value` set for the `Range` in which the time points fall.

Observe that, for time points, the computation of the transient is triggered; while for `Range`, this is not the case. `Time` takes the same options as the `Time` keyword in `Plot` in `Solve`. The specification:

```
TurningPoints ( ... (1.e-9 1.e-10) )
```

serves as shorthand for a single time-point condition:

```
TurningPoints ( ... (Condition (Time ( 1.e-9 ) ) Value= 1.e-10) )
```

#### Note:

Turning points related to optical simulations, specified using the unified interface for optical generation computation (see [Chapter 21 on page 648](#)), are described in [Optical Turning Points on page 667](#).

In the off-state of a device, the displacement current might not be resolved due to numeric precision if the time step is too small, which leads to a spiky current. In this case, you can define:

```
TurningPoints ( Condition=(...) Value=<float> Factor=<int> )
```

The parameter `Factor` controls the criterion `max_current×Value < Factor×eps`, where `eps` represents the numeric limit for the precision in use, and `max_current` is the maximum terminal current of that device.

## Chapter 4: Performing Numeric Experiments

### Transient Command

Sentaurus Device detects whether `Value` is too small to resolve the result, based on this criterion. If so, it prints a warning message and resets the time step to a more reasonable value.

By default, `Factor=0`. To solve a spiky current, set `Factor` to a positive value, for example, `Factor=10`.

---

## Ramping Physical Parameter Values

A Transient command allows parameters from the parameter file of Sentaurus Device to be ramped linearly.

The `Bias` statement (similar to the `Goal` subsection in a Quasistationary command; see [Ramping Physical Parameter Values on page 126](#)) has the form:

```
Bias {  
    [ Device = <device> ]  
    [ Material = <material> | MaterialInterface = <interface> |  
        Region = <region> | RegionInterface = <interface> ]  
    Model = <model> Parameter = <parameter> Value = <value_list>  
}
```

Specifying the device and location (material, material interface, region, or region interface) is optional. However, the model name and parameter name must always be specified. For example:

```
Transient (  
    InitialTime = 0 FinalTime = 1  
    Bias(  
        Material = "Silicon"  
        Model = DeviceTemperature Parameter = "Temperature"  
        Value = (300 at 0.1, 400 at 0.2, 450 at 0.5)  
    )  
    Bias(  
        Region = "Channel"  
        Model = DeviceTemperature Parameter = "Temperature"  
        Value = (550 at 0.6, 400 at 0.8, 300 at 0.9)  
    )  
) { Coupled { Poisson Electron Hole } }
```

For a list of parameters that can be ramped in a Transient command, see [Ramping Physical Parameter Values on page 126](#).

## Extrapolation

The `Transient` command can use the extrapolation from the solutions of the previous steps to predict the next solution. Extrapolation can be switched off (default) or on globally in the `Math` section:

```
Math { Extrapolate }
```

Alternatively, you can switch it off or on for a specific `Transient` command only:

```
Transient (
    Goal { ... }
    -Extrapolate
) { Coupled { Poisson Electron Hole } }
```

Sentaurus Device can use extrapolation in the following cases: inside a `Transient` command and between `Transient` commands.

### Inside a Transient Command

The initial guess for a given step is obtained by extrapolation from the solutions of the previous steps (if they exist).

### Between Quasistationary Commands

This extrapolation information is preserved between `Transient` commands, and it is also saved and loaded automatically by the `Save` and `Load` commands.

A `Transient` command can use this extrapolation information between the two `Transient` commands, if the following conditions are met:

1. The previous and current `Transient` commands are contiguous, that is, the final time of the previous `Transient` is equal to the initial time of the current `Transient`.
2. The values of the solution variables have not changed between the two `Transient` commands, for example, by a `Load` command.

If the extrapolation information from a previous `Transient` command can be used successfully, the following message appears in the log file:

```
Reusing extrapolation from a previous transient
```

The options to control the handling of extrapolation information described in [Table 19 on page 132](#) are available for `Transient` as well. See [Parameters of the Extrapolate Statement on page 133](#).

## Transient Ramps

As an alternative, Sentaurus Device provides `Transient` command syntax very similar to the `Quasistationary` command. This `Transient` command simplifies switching from quasistationary simulations to slow transient ones with minimal user effort. Replacing a quasistationary simulation with a slow transient is particularly useful in modeling wide-bandgap semiconductor devices, devices with trap states, breakdown simulations, and in general where convergence can be improved by switching to transient.

The transient ramp is activated by a `Quasistationary`-like command where the `Quasistationary` keyword is replaced by `Transient` and two optional parameters `InitialTime` and `FinalTime` can be used to control the ramp rate:

```
Solve { ...
    Transient (
        InitialTime = 0
        FinalTime = 1
        InitialStep = 0.01
        MaxStep = 0.1
        MinStep = 0.001
        Goal {name="gate" Voltage=5.0}
        Goal {name="drain" Current=1e-8}
    ) { Coupled {...} }
}
```

The default value for `InitialTime` is 0 or the final simulation time from a previous transient ramp simulation. The `FinalTime` default value is `InitialTime + 1` second.

When a quasistationary ramp is inserted between two transient ramps, the time for the transient ramp after the quasistationary will be reset to zero if `InitialTime` is not specified.

If the dynamic nonlocal path band-to-band model is activated, the `NonlocalPath` section can be included (see [Handling Derivatives on page 535](#)).

In the following example, the last transient biases the contact `anode` from 3 V at  $t = 0$  to 0 V at  $t = 2$  s:

```
Solve { ...
    Transient (
        FinalTime = 2
        InitialStep = 0.01
        MaxStep = 0.1
        MinStep = 0.001
        Goal {name="anode" Voltage=2.0}
    ) { Coupled {...} }

    Quasistationary(
        InitialStep = 0.01
        MaxStep = 0.1
        MinStep = 0.001
```

## Chapter 4: Performing Numeric Experiments

### Large-Signal Cyclic Analysis

```
    Goal {name="anode" Voltage=3.0}
) { Coupled {...} }

Transient (
    FinalTime = 2
    InitialStep = 0.01
    MaxStep = 0.1
    MinStep = 0.001
    Goal {name="anode" Voltage=0.0}
) { Coupled {...} }
}
```

The electrode bias time dependency is computed internally for the electrodes specified in the `Goal` section, so no time-dependent bias specification is allowed in the `Electrode` section for those contacts. `InitialStep`, `MaxStep`, and `MinStep` are like those in the Quasistationary case represented on a 0 to 1 scale. Sentaurus Device converts them internally to a scale from `InitialTime` to `FinalTime`.

This feature supports voltage, current, or charge ramps. Therefore, in the `Goal` section, only voltage, current, and charge contacts are allowed.

---

## Large-Signal Cyclic Analysis

For high-speed and high-frequency operations, devices are often evaluated by cyclic biases. After a time, device variables change periodically. This cyclic-bias steady state [3] is a condition that occurs when all parameters of a simulated system return to the initial values after one cycle bias is applied.

In fact, such a cyclic steady state is reached by using standard transient simulation. However, this is not always effective, especially if some processes in the system have very long characteristic times in comparison with the period of the signal.

For example, deep traps usually have relatively long characteristic times. A suggested approach [4] allows for significant acceleration of the process of reaching a cyclic steady state solution. The approach is based on iterative correction of the initial guess at the beginning of each period of transient simulation, using previous initial guesses and focusing on reaching a cyclic steady state. This approach is implemented in Sentaurus Device. An alternative frequency-domain approach is harmonic balance (see [Harmonic Balance on page 153](#)).

---

## Description of Method

The original method [4] is summarized. Transient simulation starts from some initial guess. A few periods of transient simulation are performed and, after each period, the change over the period of each independent variable of the simulated system is estimated (that is, the

## Chapter 4: Performing Numeric Experiments

### Large-Signal Cyclic Analysis

potential at each vertex of all devices, electron and hole concentrations, carrier temperatures, and lattice temperature if hydrodynamic or thermodynamic models are selected, trap occupation probabilities for each trap type and occupation level, and circuit node potentials in the case of mixed-mode simulation).

If  $x_n^I$  denotes the value of any variable in the beginning of the  $n$ -th period, and  $x_n^F$  denotes the same value at the end of the period, the cyclic steady state is reached when:

$$\Delta x_n = x_n^F - x_n^I \quad (16)$$

is equal to zero.

Considering linear extrapolation and that the goal is to achieve  $\Delta x_{n+1} = 0$ , the next initial guess can be estimated as:

$$x_{n+1}^I = x_n^I - \gamma \frac{x_n^I - x_{n-1}^I}{\Delta x_n - \Delta x_{n-1}} \Delta x_n \quad (17)$$

where  $\gamma$  is a user-defined relaxation factor to stabilize convergence.

As [Equation 17](#) contains uncertainty such as 0/0, especially when  $\Delta x$  is close to zero (when the solution is close to the steady state), special precautions are necessary to provide robustness of the algorithm.

Consider the derivation of [Equation 17](#) in a different fashion – near the cyclic steady state. If such a steady state exists, the initial guess  $y = x^I$  is expected to behave with time as  $y = a \exp(-\alpha t) + b$ , where  $\alpha > 0$ . It is easy to show that [Equation 17](#) gives  $x^I = b$ , that is, a desirable cyclic steady-state solution.

It follows that the ratio  $r$ :

$$r = -\frac{x_n^I - x_{n-1}^I}{\Delta x_n - \Delta x_{n-1}} \quad (18)$$

can be estimated as  $r \approx 1/(1 - \exp(-\alpha t))$ . From this, it is clear that because  $\alpha$  is positive, the condition  $r \geq 1$  must be valid. Moreover,  $r$  can be very large if some internal characteristic time (like the trap characteristic time) is much longer than the period of the cycle.

Using the definition of  $r$  from [Equation 18](#), [Equation 17](#) can be rewritten as:

$$x_{n+1}^I = x_n^I + \gamma r \Delta x_n \quad (19)$$

Although [Equation 18](#) and [Equation 19](#) are equivalent to [Equation 17](#), it is more convenient inside Sentaurus Device to use [Equation 18](#) and [Equation 19](#). Sentaurus Device never allows  $r$  to be less than 1 because of the above arguments.

To provide convergence and robustness, it is reasonable also not to allow  $r$  to become very large. In Sentaurus Device,  $r$  cannot exceed a user-specified parameter  $r_{\max}$ . You can specify different  $r_{\max}$  values for temperature, potential, and circuit node voltages. By

## Chapter 4: Performing Numeric Experiments

### Large-Signal Cyclic Analysis

default,  $r_{\max}$  is used for each equation. You can also specify the value of the parameter  $r_{\min}$ .

An extrapolation procedure, which is described by [Equation 18](#) and [Equation 19](#), is performed for every variable of all the devices, in each vertex of the mesh. Instead of densities, which can spatially vary over the device by many orders of magnitude, the extrapolation procedure is applied to the appropriate quasi-Fermi potentials.

For the trap equations, extrapolation can be applied either to the trap occupation probability  $f_T$  (the default) or, optionally, to the ‘trap quasi-Fermi level,’  $\Phi_T = -\ln((1-f_T)/f_T)$ . The cyclic steady state is supposedly reached if the following condition is satisfied:

$$\frac{\Delta x}{x + x_{\text{ref}}} < \varepsilon_{\text{cyc}} \quad (20)$$

Values of  $x_{\text{ref}}$  are the same as `ErrRef` values of the `Math` section. For every object  $o$  of the simulated system (that is, every variable of all devices), an averaged value  $r_{\text{av}}^o$  of the ratio  $r$  is estimated and can be optionally printed. Estimation of  $r_{\text{av}}^o$  is performed only at such vertices of the object, where the condition:

$$\frac{\Delta x}{x + x_{\text{ref}}} < \frac{\varepsilon_{\text{cyc}}}{f} \quad (21)$$

is fulfilled, that is, the same condition as [Equation 20](#), but with a possibly different tolerance  $\varepsilon_{1\text{cyc}} = \varepsilon_{\text{cyc}}/f$ .

You can specify the cyclic norm file name in the `File` section of the command file to store the cyclic norm as a function of the simulation time and the number of cyclic cycles for visualization of cyclic convergence.

The following extrapolation procedures are allowed:

1. Use of averaged extrapolation factors for every object. This is the default option.
2. Use of the factor  $r$  independently for every mesh vertex of all objects. If for some reason, the criterion in [Equation 21](#) is already reached, the value of factor  $r$  is replaced by the user-defined parameter  $r_{\min}$ . The option is activated by the keyword `-Average` in the `Extrapolate` statement inside the `Cyclic` specification.
3. The same as Step 2, but for the points where [Equation 21](#) is fulfilled, the value of factor  $r$  is replaced by the averaged factor  $r_{\text{av}}^o$ . The option is activated by the keyword `-Average` in the `Extrapolate` statement inside the `Cyclic` specification, accompanied by the specification of the parameter  $r_{\min} = 0$ .
4. You can limit the extrapolation factor for temperature equations (lattice, carrier temperatures), circuit node voltage, and electrostatic potential separately from carrier concentrations. For details about the associated keywords, see [Table 341 on page 1720](#).
5. You can limit the cyclic extrapolation to a few cyclic cycles beyond which Sentaurus Device automatically switches to transient simulations. This allows the use of a mixed

## Chapter 4: Performing Numeric Experiments

### Large-Signal Cyclic Analysis

cyclic-transient methodology. This option can avoid the solution from oscillating around steady state.

6. You can specify a slightly modified extrapolation method using the keyword `CyclicExtrapolationMethod`. By default, Sentaurus Device uses [Equation 17](#) for extrapolation (`CyclicExtrapolationMethod=0`). However, in certain cases, setting `CyclicExtrapolationMethod=1`, which uses a slightly different formulation, can help with cyclic convergence.
7. For mixed-mode simulations, it has been observed that extrapolation applied to carrier concentrations often results in convergence issues. You can choose to exclude any field variable, that is,  $\{\psi, n, p, QP, T1, Tc\}$ , from cyclic extrapolation. The `IgnoreElectronPDE` option excludes electron density from cyclic extrapolation. Keywords relevant to this feature are explained in [Table 341 on page 1720](#).
8. You can ignore a particular *region* for extrapolation. If specified in the command file, then none of the fields in the user-specified region is extrapolated using the cyclic extrapolation scheme. Fields in all other regions continue to be extrapolated cyclically.

---

## Using Cyclic Analysis

Cyclic analysis is activated by specifying the parameter `Cyclic` in the parameter list of the `Transient` statement. `Cyclic` is a complex structure-like parameter and contains cyclic options and parameters in parentheses:

```
Transient(
    InitialTime=0 FinalTime=2.e-7 InitialStep=2.e-14 MinStep=1.e-16
    Cyclic( <cyclic-parameters> )
) { ... }
```

With sub-options to the optional parameter `Extrapolate` to the `Cyclic` keyword, details of the cyclic extrapolation procedure are defined. [Table 341 on page 1720](#) lists all options for `Cyclic`.

An example of a `Transient` command with `Cyclic` specification is:

```
Transient(
    InitialTime=0 FinalTime=2.e-7 InitialStep=2.e-14 MinStep=1.e-16
    Cyclic( Period=8.e-10 StartingPeriod=4
        Accuracy=1.e-4 RelFactor=1
        Extrapolate (Average Print MaxVal=50) )
) { ... }
```

### Note:

The value of the parameter `Period` in the `Cyclic` statement must be divisible by the period of the bias signal. A periodic bias signal must be specified elsewhere in the command file.

## Small-Signal AC Analysis

An `ACCoupled` solve section is an extension of a `Coupled` section with an extra set of parameters allowing small-signal AC analysis. [Table 338 on page 1717](#) describes these parameters. In general, an `ACCoupled` is used in mixed mode. [AC Simulation on page 1191](#) provides technical background information for the method.

### AC Analysis in Mixed-Mode Simulations

AC analysis computes the frequency-dependent admittance matrix  $Y$  between circuit nodes of the specified electrical system.

For a given excitation frequency  $\nu$ , it describes the equivalent small-signal model by:

$$\delta I = Y \delta V \quad (22)$$

where  $\delta V$  and  $\delta I$  are the vectors of (complex-valued) voltage and current excitations at selected nodes, respectively. The admittance matrix can be represented as:

$$Y = A + i2\pi\nu C \quad (23)$$

by the real-valued conductance matrix  $A$  and capacitance matrix  $C$ .

Within an `ACCoupled` section, you must specify the frequencies of interest and the circuit nodes considered in the admittance matrix. Furthermore, you might have to exclude some circuit instances from the given electrical system to describe the proper AC system:

- `StartFrequency`, `EndFrequency`, `NumberOfPoints`, `Linear`, and `Decade`: Select the frequencies at which the AC analysis is performed.
- `Frequency`: Specify a list of frequencies at which the AC analysis is performed.
- `ACEExtract`: Specify an `ACCoupled`-specific file-name prefix for the extraction file, where the admittance matrices will be stored. If not specified, the `ACCoupled` writes into the global extraction file (given by `ACEExtract` in `File`, which defaults to the extraction file `extraction_ac_des.plt`). The extraction file contains the frequency, the voltages at the nodes, and the entries of the matrices  $A$  (denoted by `a`) and  $C$  (denoted by `c`).
- `ACPlot`: Invoke device plots containing AC response functions (solution variables and current densities). Here, you specify a file-name prefix, which generates a corresponding plot file for each AC node voltage excitation and frequency.
- `Exclude`: Specify a list of system instances that should not be part of the AC system.
- `Node`: Specify the list of AC nodes considered in the admittance matrix. For admittance matrix computations, a nonempty node list must be specified.

## Chapter 4: Performing Numeric Experiments

### Small-Signal AC Analysis

#### Note:

If both `Frequency` and either one of `StartFrequency`, `EndFrequency`, or `NumberOfPoints` are defined, then the value defined in `Frequency` is used.

The `Exclude` list is used to remove a set of circuit or physical devices from the AC analysis. Typically, the power supply is removed so as not to short-circuit the AC analysis, but the list can also be used to isolate a single device from a whole circuit.

#### Note:

The system analyzed consists of the equations specified in the body of the `ACCoupled` statement, without the instances removed by the `Exclude` list. The `Exclude` list only specifies instances, therefore, all equations of these instances are removed.

The `ACCompute` option controls AC or noise analysis performances within a Quasistationary or Transient ramp. The parameters in `ACCompute` are identical to the parameters in the `Plot` and `Save` commands (see [Table 345 on page 1727](#)). For example:

```
Quasistationary (...) {
    ACCoupled (... {
        ACCompute (Time = (0; 0.01; 0.02; 0.03; 0.04; 0.05)
                    Time = (Range = (0.9 1.0) Intervals = 4)
                    Time = (Range = (0.1 0.2); Range = (0.7 0.8))
                    When (Node = in Voltage = 1.5))
    {...}
}
```

In this example, an AC analysis is performed only for the time points:

```
t = 0, 0.01, 0.02, 0.03, 0.04, 0.05
t = 0.9, 0.925, 0.95, 0.975, 1.0
```

and for all time points in the intervals [0.1, 0.2] and [0.7, 0.8]. In addition, an AC analysis is triggered whenever the voltage at the node `in` reaches the 1.5 V threshold.

If the AC or noise analysis is suppressed by the `ACCompute` option, an `ACCoupled` statement behaves like an ordinary `Coupled` statement.

The Quasistationary or Transient ramp that contains the `ACCoupled` also can contain a `CurrentPlot` specification (see [When to Write to the Current File on page 162](#)) to select the steps at which current output should occur.

The `CurrentPlot` and an `ACCompute` statement that the `ACCoupled` statement might contain are completely independent.

## Example

This example illustrates AC analysis of a simple device. A 1D resistor is connected to ground (through resistor `to_ground`) and to a voltage source `drive` at a reverse bias of

## Chapter 4: Performing Numeric Experiments

### Small-Signal AC Analysis

–3 V. After calculating the initial voltage point at –3 V, the left voltage is ramped to 1 V in 0.1 V increments. The AC parameters between nodes `left` and `right` are calculated at frequencies  $f = 10^3$  Hz,  $10^4$  Hz,  $10^5$  Hz, and  $10^6$  Hz. The circuit element `drive` and `to_ground` are excluded from the AC calculation.

By including circuits, complete Bode plots can be performed:

```
Device "Res" { ...
    Electrode {{Name=anode    Voltage=-3 resist=1}
                {Name=cathode Voltage= 0 resist=1}}
    }
    File {Grid = "resist.tdr"}
    Physics {...}
}

System {
    "Res" "1d" (anode="left" cathode="right")
    Vsource_pset drive("left" "right"){dc = -3}
    Resistor_pset to_ground ("right" 0){resistance=1}
}

Math {
    Method=Blocked SubMethod=Super # 1D, 2D default solvers for Coupled
    ACMETHOD=Blocked ACSUBMETHOD=Super # 1D, 2D default solvers for AC
        # analysis
    NoAutomaticCircuitContact
}

Solve { ...
    ACCoupled (
        StartFrequency=1e3 EndFrequency=1e6 NumberOfPoints=4 Decade
        Node("left" "right")
        Exclude(drive to_ground)
        ACMETHOD=Blocked ACSUBMETHOD("1d")=ParDiSo
    ) { Poisson Electron Hole Contact Circuit }
    Quasistationary ( ...
        Goal{Parameter=drive.dc Value=1}
    ) {
        ACCoupled(
            StartFrequency=1e3 EndFrequency=1e6 NumberOfPoints=4 Decade
            Node("left" "right")
            Exclude(drive to_ground)
            ACMETHOD=Blocked ACSUBMETHOD("1d")=ParDiSo
        ) { Poisson Electron Hole Circuit Contact }
    }
}
```

## AC Analysis in Single-Device Mode

As previously described, AC analysis requires in general a mixed-mode simulation, that is, a `System` section must be defined. For single-device simulations, a simple AC system is constructed for you if you use `ImplicitACSystem` in the global `Math` section.

This implicit AC system is built internally and is essentially invisible to users. The implicit AC system has the following properties:

- For each voltage-controlled electrode of the device under test (DUT), an electrical circuit node is constructed and connected to the device contact. For other contact types, no nodes are constructed.
- Each implicit node is connected with a system instance describing the stationary and (possibly) transient voltage boundary conditions. Data provided for the device contacts is transferred implicitly to the system instances.

In the case of an implicit AC system generation, the `ACCoupled` provides slightly different default behavior:

- `Node`: If no AC nodes are specified, all implicit system nodes are used as AC nodes. Specified device contact names are interpreted implicitly as the corresponding node name.
- `Exclude`: If no exclude instances are specified, all implicit system instances attached to the AC nodes are excluded. Specified device contact names are interpreted as the corresponding system instance name.

Additional remarks:

- `Goal` statements in `Quasistationary`: `Goal` statements for the electrode-voltage values for the device are interpreted as `Goal` statements for the implicit system instance connected to the corresponding node of the electrode.
- The implicit AC system is extracted before the `Solve` section is executed. Therefore, solve statements that change the mode of the contact type are not supported.

### Example

```
Math {
    ImplicitACSystem           * build implicit AC system
}

Electrode { ...
    { Name="c1" Voltage=1 Voltage=(1 at 0. , 2. at 1.e-8) ... }
    { Name="c2" Current=0. }   * contact without implicit node connection
}

Solve { ...
    Quasistat ( ...
```

## Chapter 4: Performing Numeric Experiments

### Harmonic Balance

```
Goal { Name= "c1" Voltage=2" }           * contact goals are mapped
      * onto instances
) { Coupled {...} }

* AC analysis: use all implicit AC nodes
ACCoupled (
    StartFrequency=1.e6 EndFrequency=1.e9 NumberOfPoints=4 Decade
    ACEExtract="AC1" ) {...}
* AC analysis: use only one AC node
ACCoupled (
    StartFrequency=1.e6 EndFrequency=1.e9 NumberOfPoints=4 Decade
    Node ( "c1" )
    ACEExtract="AC2" ) {...}
}
```

---

## Optical AC Analysis

Optical AC analysis calculates the quantum efficiency as a function of the frequency of the optical signal. This technique is based on the AC analysis technique, and provides real and imaginary parts of the quantum efficiency versus the frequency.

To start optical AC analysis, add the keyword `Optical` in an `ACCoupled` statement. For example:

```
ACCoupled ( StartFrequency=1.e4 EndFrequency=1.e9
            NumberOfPoints=31 Decade Node(a c)
            Optical Exclude(v1 v2) )
            { poisson electron hole }
```

For details, see [Optical AC Analysis on page 770](#).

---

## Harmonic Balance

Harmonic balance (HB) analysis is a frequency domain method to solve periodic or quasi-periodic, time-dependent problems. Compared to transient analysis (see [Transient Command on page 138](#)), HB is computationally more efficient for problems with time constants that differ by many orders of magnitude. Compared to AC analysis (see [Small-Signal AC Analysis on page 149](#)), the periodic excitation is not restricted to infinitesimally small amplitudes. An alternative to HB for the periodic case is cyclic analysis (see [Large-Signal Cyclic Analysis on page 145](#)).

**Note:**

Harmonic balance is not supported for traps (see [Chapter 17 on page 543](#)) or ferroelectrics (see [Chapter 29 on page 905](#)).

## Chapter 4: Performing Numeric Experiments

### Harmonic Balance

The time-dependent simulation problems take the form:

$$\frac{d}{dt}q[r, u(t, r)] + f[r, u(t, r), w(t, r)] = 0 \quad (24)$$

where  $f$  and  $q$  are nonlinear functions that describe the circuit and the devices,  $u$  is the vector of solution variables, and  $w$  is a time-dependent excitation.

Assuming  $w$  is quasi-periodic with respect to  $f = (\hat{f}_1, \dots, \hat{f}_M)^T$ , the vector of the positive base frequencies  $f_m$ , and  $H = (H_1, \dots, H_M)$ , the vector of nonnegative maximal numbers of harmonics  $H_m$ , the solution  $u$  is approximated by a truncated Fourier series:

$$u(t) = U_0 + \sum_{-\underline{H} \leq h \leq \bar{H}} U_h \exp(i\omega_h t) \quad (25)$$

where  $\omega_h = 2\pi h \cdot \hat{f}$ . In Fourier space, [Equation 24](#) becomes:

$$L(U) := i\Omega Q(U) + F(U) = 0 \quad (26)$$

where  $\Omega$  is the frequency matrix, and  $F$  and  $Q$  are the Fourier series of  $f$  and  $q$ . The function  $L$  depends nonlinearly on  $U$  and, therefore, solving [Equation 26](#) requires a nonlinear (Newton) iteration. For more details about the numerics of harmonic balance, see [Harmonic Balance Analysis on page 1194](#).

---

## Modes of Harmonic Balance Analysis

Sentaurus Device supports two different modes to perform harmonic balance simulations: the MDFT mode and the SDFT mode.

### MDFT Mode

The MDFT mode is suitable for multitone analysis and one-tone analysis, and is enabled by the `MDFT` option in the `HB` section of the global `Math` section. It uses the multidimensional Fourier transformation (MDFT) to switch between the frequency and time domain of the system.

This mode requires compact models defined by the compact model interface (CMI), which support assembly routines in the frequency domain, that is, the CMI-HB-MDFT function set as described in [Compact Models User Guide](#), Analytical Description of CMI Models.

Sentaurus Device provides a basic set of compact models that support this functional behavior (see [Compact Models User Guide](#), CMI Models With Frequency-Domain Assembly). Standard SPICE models are not supported in this mode.

## Chapter 4: Performing Numeric Experiments

### Harmonic Balance

## SDFT Mode

The SDFT mode supports only one-tone HB analysis. The mixed-mode circuit might contain SPICE models and CMI models that do not provide the CMI-HB-MDFT functional set. It is used if the MDFT mode is disabled.

---

## Performing Harmonic Balance Analysis

Harmonic balance simulations are enabled through the keyword `HBCoupled` in the `Solve` section. The syntax of `HBCoupled` is the same as for `Coupled` (see [Coupled Statement on page 191](#)). In addition, `HBCoupled` supports the options summarized in [Table 344 on page 1725](#).

### Note:

Not all `HBCoupled` options are supported by both modes.

The `Tone` option is mandatory, as no default values are provided.

Specifying several `Tone` options in MDFT mode enables multitone analysis. The base frequencies for the analysis can be given by explicit numeric constants or can refer to the frequencies of time-dependent sources in the `System` section.

An example of a two-tone analysis is:

```
Math {
    HB { MDFT }                                * enable MDFT mode
    ...
}
System {
    ...
    sd_tb_vsource2_pset "va" ( na 0 )          * two-tone voltage source
    { dc = 1.0 freq = 1.e9 mag = 5.e-3 phase = -90.
      freq2 = 1.e3 mag2 = 1.e-3 phase2 = -90. }

    HBPlot "hbplot" ( v(na) i(va na) ... )   * HB circuit output
}
Solve {
    * solve the DC problem
    ...
    Coupled { Poisson Electron Hole }

    * solve the HB problem
    HBCoupled ( * two-tone analysis
        Tone ( Frequency = "va"."freq" NumberOfHarmonics = 5 )
        Tone ( Frequency = "va"."freq2" NumberOfHarmonics = 1 )
        Initialize = DCMode
        Method = ILS
        GMRES ( Tolerance = 1.e-2 MaxIterations = 30 Restart = 30 )
```

## Chapter 4: Performing Numeric Experiments

### Harmonic Balance

```
    ) { Poisson Electron Hole }
}
```

The base frequencies  $\hat{f}_1$  and  $\hat{f}_2$  in this example are taken from the time-dependent voltage source `va`.

`HBCoupled` can be used as a top-level `Solve` statement, or within `Plugin`, `QuasiStationary`.

#### Note:

If a `QuasiStationary` controls other `Solve` statements besides a single `HBCoupled`, step reduction due to convergence problems works correctly only when no `HBCoupled` in the failing step has converged before the statement that caused the failure is run.

## Solve Spectrum

The spectrum to be solved is determined by the specified tones. For different `HBCoupled` statements, the number of tones and their number of harmonics are allowed to change (where the  $m$ -th tone is identified with the  $m$ -th tone of the next spectrum, regardless of the value of its frequency, that is, the order of tones is significant).

For some applications, you might be interested only in a few spectrum components or you might want to reduce the computational burden in the Newton process. For such situations, you can reduce the solve spectrum (only for the MDFT mode) by applying spectrum truncation.

This is performed by specifying a list of spectrum (multi-)indices by using `SolveSpectrum` in the `HB` section of the global `Math` section, and referencing to this spectrum in the `HBCoupled` statement. For example:

```
Math {
    ...
    HB {
        ...
        SolveSpectrum ( Name = "sp1" ) { (0 0) (1 0) (0 1) (2 1)
                                         (2 -1) } }
    }
    ...
}
Solve {
    ...
    HBCoupled ( ... SolveSpectrum = "sp1" ) { ... }
}
```

where, for example, the multi-index (2 -1) corresponds to the intermodulation frequency  $2f_1 - 1f_2$ .

## Convergence Parameters

The convergence behavior of `HBCoupled` can be analyzed and influenced independently of specifications for the convergence of `Coupled` `solve` statements. Some parameters can be set exclusively in the `HB` section of the global `Math` section (see [Table 220 on page 1607](#)),

## Chapter 4: Performing Numeric Experiments

### Harmonic Balance

others can be set exclusively in the `HBCoupled` statement (see [Table 344 on page 1725](#)), and some can be set in both places.

Specifying `LogDensity` allows you to use transformed density variables in the HB analysis. Instead of using the carrier density variables directly, the logarithm of internally scaled carrier densities are used as solution variables in the HB Newton algorithm. This improves sometimes the convergence of the analysis. You can set this flag only in the global `Math` section, and the variable transformation applies to all `HBCoupled` statements.

The parameter `LogTemperature` works completely analogously to `LogDensity`, but here the variable transformation is applied to the (internally scaled) lattice and carrier temperature variables. This option requires the novel update error computation.

The parameter `sdft_update_norm_R` allows you to change between the update error computation (compatible with Version R-2020.09) and an improved novel variant. For linear variables, the differences are very small, but for logarithmic variables (that is, usage of `LogDensity` or `LogTemperature`), the novel update error computation deviates significantly and is more consistent. Therefore, it is highly recommended to switch off the old variant by specifying `-sdft_update_norm_R` in the `HB` section of the global `Math` section.

`CNormPrint` is used to print the residuum, the update error, and the number of corrections (the number of grid points where a correction of computed sample points is necessary) in each Newton step for all solved equations of all instances. This information can be used to adjust the numeric convergence parameters.

The `Derivative` flag in `HBCoupled` overwrites the `Derivative` flag of the `Math` section and determines whether all derivatives are included in the HB Newton process.

With `RhsScale` and `UpdateScale`, you can scale the residuum and the update error of individual equations, respectively, which are used as Newton convergence criteria. This is often necessary for the electron and hole continuity equations, and the values differ for different applications and devices.

The parameters `ValueMin` and `ValueVariation` are used for positive solution variables to specify the allowed minimum value in the time domain, and the corresponding ratio of maximum and minimum values, respectively. The number of corrections (given by `CNormPrint`) indicates how many grid points violate the specified bounds.

## Additional Remarks

Typical HB simulations are mixed-mode simulations specified in the `System` section (see [Chapter 3, Mixed-Mode Simulations](#)).

You can add the option `SystemTopology(Print)` in the global `Math` section, which generates a description and a simplified analysis of the present circuit.

The analysis extracts sets of circuit nodes that are DC-conductively connected. If there are more than one such connectivity sets, then the physical problem might be underdetermined

## Chapter 4: Performing Numeric Experiments

### Harmonic Balance

and cannot be solved. For example, if there is one connectivity component consisting of only one node, and if this node is not set to a fixed value, then this node is floating and its voltage cannot be determined. In practice, you then must revise your circuit and connect the node conductively with other parts.

Note that, so far, only `Capacitor` instances are considered to be non-DC conductive, and physical devices are considered to be conductive between all their used electrodes.

---

## Harmonic Balance Analysis Output

All frequency-domain output data refers to the one-sided Fourier series representation for real-valued quantities, that is, it refers to  $\underline{U}_h$  of:

$$\begin{aligned} u(t) &= \tilde{U}_0 + \sum_{\underline{h} \in K^+} \text{Re}(\tilde{\underline{U}}_{\underline{h}} \exp(i\omega_{\underline{h}} t)) \\ &= \tilde{U}_0 + \sum_{\underline{h} \in K^+} \text{Re}(\tilde{\underline{U}}_{\underline{h}}) \cos(\omega_{\underline{h}} t) - \text{Im}(\tilde{\underline{U}}_{\underline{h}}) \sin(\omega_{\underline{h}} t) \end{aligned} \quad (27)$$

where the sum is taken over all multi-indices  $\underline{h}$ , which results in a positive frequency  $\omega_{\underline{h}} = \underline{h} \cdot \underline{\omega}$ .

## Circuit Currents and Voltages

The keyword `HBPlot` in the `System` section allows you to plot circuit quantities. The syntax of `HBPlot` is identical to that of `Plot` in the `System` section (see [Plotting Quantities on page 109](#)). For both SDFT and MDFT modes, two output files are generated for the circuit quantities: `HB_F` for the frequency domain output, and `HB_T` for the time domain output. Output quantities for all bias points are combined into such output files in a way similar to AC analysis.

If conflicts impede information being saved in the same `.plt` file, then Sentaurus Device closes the current file, automatically modifies the file name by inserting an incrementer (for example, `HB_F_1` and `HB_T_1`), and prints a warning.

Output files for individual devices are suppressed as HB information is usually related to circuit nodes.

To deactivate the new output format, define:

```
Math { -NewHBFilenames }
```

The old output files (`.plt` files per converged nonlinear solve) are not plotted by default, and you must activate this explicitly by specifying:

```
Math { HBplotFilePerNewton }
```

## Solution Variables

Plotting solution variables is implicitly performed if the HB section in the `Math` section is present. Sentaurus Device plots the magnitude and phase of the  $U_h$  coefficients of [Equation 27](#), with the prefixes `MagHB` and `PhaseHB` to the variable names. For the SDFT mode, a suffix `_C<i>` with component `<i>` is added to the names. For the MDFT mode, a suffix corresponding to the intermodulation frequencies `_C<i,j,k...>` is added to the names. To control the output quantities, you can define:

```
Plot { ...
    HBPlot ( <int>|order=<int> dataname[<list>] )
}
```

where `<int>` or `order=<int>` defines the order for plotting. For the SDFT mode, `order` is equal to the number of harmonics to be plotted. For the MDFT mode, it plots the results of intermodulation frequencies up to the integer defined by `order`. For example, for `order=3`, in two-tone simulations (frequencies  $f_2 > f_1$ ), Sentaurus Device plots the datasets corresponding to  $f_1, f_2, 2xf_1, 2xf_2, f_1+f_2, f_2-f_1, 3xf_1, 3xf_2, 2xf_1+f_2, 2xf_2+f_1, 2xf_1-f_2, 2xf_2-f_1$ , as long as the resulting frequency is positive. If `order` is not defined, then the results of all harmonics used in the simulation are plotted. You can set `order=0` to suppress the HB datasets from the TDR files.

The keyword `dataname[<list>]` defines a list of datasets to be plotted. If `dataname` is not defined, then all related solution variables are plotted; otherwise, only the defined ones are plotted.

If options for `HBPlot` are not defined, then the results of all harmonics of all related solution variables are plotted, which is the default case.

## Additional Plot Variables

If you plot solution variables by using `HBPlot` in the `Plot` section, the simulator automatically provides additional variables that describe the actual HB state. This information can be used to analyze the actual HB state and might be useful to improve the simulation mesh and additional simulation parameters. They are described here for the specific variable `eDensity`, but apply analogously to other solution variables:

- `MinTDeDensity`, `MaxTDeDensity`: Minimal and maximal values of the solution variable within the period of the signal
- `DiffTDElectrostaticPotential`: The difference `MaxTD - MinTD` of the solution variable (not available for densities and temperatures)
- `RatioTDeDensity`: The quotient `MaxTDeDensity/MinTDeDensity` (available for densities and temperatures)
- `VarHBeDensity`: A measure for the width of the significant spectrum

## Chapter 4: Performing Numeric Experiments

### References

It provides some information about how many harmonics are needed to describe the local solution variable. The value is close to zero if no harmonics have a significant size; it is approximately one if only the fundamental harmonic contributes to the spectrum (that is, the local signal is essentially linear), and it exceeds one if higher harmonics become relevant (that is, indicating local distortion). Experimental results have shown that values of approximately  $H/2$  (where  $H$  is the number of used harmonics) and greater indicate that aliasing might come into play, which is often the root cause for subsequent convergence issues.

## Output of Individual HBCoupled Statements

By using `HBPlotFilePerNewton`, each converged `HBCoupled` statement plots its results into separate files.

The device results (contact currents and voltages, temperatures, and heat component) in the time domain and frequency domain are plotted into individual files.

The file names for the time domain contain a component `Tdom`; the file names for the frequency domain contain a component `Hdom`.

Circuit currents and voltages are plotted as previously described, referring to the `HBPlot` statements in the `System` section, but now individual files are generated per converged `HBCoupled` statements.

---

## Application Notes

Note that:

- Convergence: Typically, the nonlinear convergence improves with an increasing number of harmonics  $H$  for one-tone HB simulations.
- Linear solvers: Benefiting both memory requirements and simulation time, you can use the iterative linear solver GMRES for most simulations. Only for very small problems, in terms of grid size and number of harmonics, will the direct solver method be sufficient.

---

## References

- [1] For information about Perl regular expression syntax, go to [https://www.boost.org/doc/libs/1\\_72\\_0/libs/regex/doc/html/boost\\_regex/syntax/perl\\_syntax.html](https://www.boost.org/doc/libs/1_72_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html).
- [2] R. J. G. Goossens *et al.*, “An Automatic Biasing Scheme for Tracing Arbitrarily Shaped *I-V* Curves,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 310–317, 1994.

## Chapter 4: Performing Numeric Experiments

### References

- [3] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*, Boston: Kluwer Academic Publishers, 1990.
- [4] Y. Takahashi, K. Kunihiro, and Y. Ohno, "Two-Dimensional Cyclic Bias Device Simulator and Its Application to GaAs HJFET Pulse Pattern Effect Analysis," *IEICE Transactions on Electronics*, vol. E82-C, no. 6, pp. 917–923, 1999.

# 5

## Simulation Results

---

*This chapter describes the output of Sentaurus Device, which contains the simulation results.*

This chapter describes only the most important output that Sentaurus Device provides. Many more specific output files exist. They are described in context in other parts of the user guide.

---

### Current File

Sentaurus Device provides several forms of output. Most importantly, the current file contains the terminal characteristics obtained during the numeric experiment. Plot files allow you to visualize device internal quantities, and thereby provide information not available in real experiments. Other output, such as the log file, allows you to investigate the simulation procedure itself and provides an important tool to understand and resolve problems with the simulation itself.

---

### When to Write to the Current File

By default, currents are output after each iteration in a `Plugin`, `Quasistationary`, or `Transient` command. This behavior can be modified by a `CurrentPlot` statement in the body of these commands. The `CurrentPlot` statement in the `Solve` section provides full control over the plotting of device currents and circuit currents. If a `CurrentPlot` statement is present, it determines exactly which points are written to the current file. Sentaurus Device can still perform computations for intermediate points, but they are not written to the file.

**Note:**

Do not confuse the `CurrentPlot` statement in the `Solve` section with the `CurrentPlot` section described in [Tracking Additional Data in the Current File on page 165](#).

## Chapter 5: Simulation Results

### Current File

The syntax of the CurrentPlot statement is:

```
CurrentPlot (<parameters-opt>) {<system-opt>}
```

Both <parameters-opt> and <system-opt> are optional and can be omitted.

<parameters-opt> is a space-separated list, which can consist of the following entries:

```
Time = (<entry> ; <entry> ; <entry> ; )
```

The list of time entries enumerates the times for which a current plot is requested. The entries are separated by semicolons. A time entry can have these forms:

- floating-point number  
The time value for which a current plot is requested.
- Range = (a b)  
This option specifies a free plot range between a and b. All the time points in this range are plotted.
- Range = (a b) Intervals = n  
This option specifies n intervals in the range between a and b. In other words, these plot points are generated:

$$t = a, t = a + \frac{b-a}{n}, \dots, t = b - \frac{b-a}{n}, t = b \quad (28)$$

```
Iterations = (<int>; <int>; <int>; )
```

The list of integers specifies the iterations for which a plot is required. This option is available for the `Plugin` command.

```
IterationStep=<int>
```

This option requests a current plot every n iterations. It is available for the `Plugin` command.

```
When (<when_condition>)
```

A `When` option can be used to request a current plot whenever a condition has been met. This option works in the same way as in a `Plot` or `Save` command (see [Table 345 on page 1727](#)).

<system-opt> is a space-separated list of devices. If <system-opt> is not present, then Sentaurus Device plots all device currents and the circuit (in mixed-mode simulations). If <system-opt> is present, then only the currents of the given devices are plotted. The keyword `Circuit` can be used to request a circuit plot.

## Chapter 5: Simulation Results

### Current File

## Example: CurrentPlot Statements

A CurrentPlot statement by itself creates a current plot for each iteration:

```
Quasistationary (
    InitialStep=0.2 MinStep=0.2 MaxStep=0.2
    Goal { Name="drain" Voltage=0.5 }
    { Coupled { Poisson Electron Hole }
    CurrentPlot }
```

If no current plots are required, then a current plot for the ‘impossible’ time  $t = -1$  can be specified:

```
Quasistationary (
    InitialStep=0.2 MinStep=0.2 MaxStep=0.2
    Goal { Name ="drain" Voltage=0.5 })
    { Coupled { Poisson Electron Hole }
    CurrentPlot ( Time = (-1)) }
```

In this example, the currents of the device nmos are plotted for  $t = 0, 10^{-8}$ , and  $10^{-7}$ :

```
Transient ( MaxStep=1e-8 InitialTime=0 FinalTime=1e-6 )
    { Coupled { Poisson Circuit }
    CurrentPlot (Time = (0;1e-8;1e-7)) { nmos } }
```

This CurrentPlot statement produces 11 equidistant plot points in the interval  $0, 10^{-5}$ :

```
Transient ( MaxStep = 1e-8 InitialTime=0 FinalTime=1e-5 )
    { Coupled { Poisson Circuit }
    CurrentPlot (Time =(range = (0;1e-5) intervals = 10)) }
```

In this example, a current plot for iteration 1, 2, 3, and for every tenth iteration is specified:

```
Plugin { Poisson Electron Hole
    CurrentPlot ( Iterations = (1; 2; 3) IterationStep = 10 ) }
```

A CurrentPlot statement can also appear at the top level in the Solve section. In this case, the currents are plotted when the flow of control reaches the statement.

A CurrentPlot statement is also recognized within a Continuation command. In this case, the time in the CurrentPlot statement corresponds to the arc length in the Continuation command. However, only free plot ranges are supported.

---

## NewCurrentPrefix Statement

By default, Sentaurus Device saves all current plots in one file (as defined by the variable Current in the File section). This behavior can be modified by the NewCurrentPrefix keyword in the Solve section:

```
NewCurrentPrefix = <string>
```

## Chapter 5: Simulation Results

### Current File

This keyword appends the specified prefix to the default current file name, and all subsequent `Plot` statements are directed to the new file. You can specify `NewCurrentPrefix` multiple times in the `Solve` section. For example:

```
Solve {
    Circuit
    Poisson
    NewCurrentPrefix = "pre1"
    Coupled {Poisson Electron Hole Contact Circuit}
    NewCurrentPrefix = "pre2"
    Transient (
        MaxStep= 2.5e-6 InitialStep=1.0e-6
        InitialTime=0.0 FinalTime=0.0001
        Plot {range=(10e-6,40e-6) Intervals=10}
    )
    {Coupled {Poisson Electron Hole Contact Circuit}}
}
```

In this example, the current files specified in the `File` and `System` sections contain the results of the `Circuit` and `Poisson` solves. The results of the `Coupled` solution are saved in a new current file with the same name but prefixed with `pre1`. The last current file contains the results of the `Transient` solve with the prefix `pre2`.

#### Note:

The file names for current plots defined in the `System` section and the plot files of AC analyses are also modified by `NewCurrentPrefix`.

---

## Tracking Additional Data in the Current File

The `CurrentPlot` section on the top level of the command file is used to include selected parameter values and mesh data into the current plot file (.plt).

The following options are available:

- You can list the required quantities directly in the `CurrentPlot` section (see [CurrentPlot Section on page 166](#)).
- You can use a Tcl formula (see [Tcl Formulas on page 170](#)).
- You can use a current plot PMI (see [Current Plot File on page 1439](#)).
- You can use the current plot Tcl interface, which represents an alternative to the current plot PMI (see [Current Plot File on page 1491](#)).

## CurrentPlot Section

Sentaurus Device can add the scalar data listed in [Appendix F on page 1515](#) to the current plot file.

Data can be plotted according to coordinates. A coordinate is given as one to three (depending on device dimensions) numbers in parentheses. When plotting according to coordinates, the plotted values are interpolated as required.

Furthermore, it is possible to output averages, integrals, maximum, and minimum of quantities over specified domains. To do this, specify the keyword `Average`, `Integrate`, `Maximum`, or `Minimum`, respectively, followed by the specification of the domain in parentheses. A domain specification consists of any number of the following:

- Region specification: `Region=<string>`
- Material specification: `Material=<string>`
- Region interface specification: `RegionInterface=<string>`
- Material interface specification: `MaterialInterface=<string>`
- Any of the keywords `Semiconductor`, `Insulator`, and `Everywhere`, which match all semiconductor regions, all insulator regions, or the entire device, respectively
- Window specification: `Window[ (x1 y1 z1) (x2 y2 z2) ]`
- Well specification: `DopingWell(x1 y1 z1)` (for the definition of the doping well boundary, see [Initial Guess for Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells on page 234](#))

**Note:**

If periodic boundary conditions are activated for doping wells, then any `CurrentPlot` operation is performed on all connected wells that are adjoined by the periodic BC. The supported periodic BCs are mortar and Robin PBC.

The average, integral, maximum, and minimum are applied to all of the specified parts of the device. Multiple specifications of the same part of the device are insignificant. In addition, `Name=<string>` is used to specify a name under which the average, integral, maximum, and minimum are written to the `.plt` file. (By default, the name is automatically obtained from a concatenation of the names in the domain specification, which yields impractically long names for complicated specifications.)

For maximum and minimum, Sentaurus Device can write coordinates where the maximum and minimum occur to the `.plt` file.

## Chapter 5: Simulation Results

### Current File

In the case of average and integral, Sentaurus Device can write the coordinates ( $x$ ,  $y$ ,  $z$ ) of the centroid of a data field  $f$  defined:

$$x = \frac{\int f(x, y, z) dV}{\int f(x, y, z) dV} \quad (29)$$

where the integration covers the specified domain. The values of  $y$  and  $z$  are defined similarly. Output of coordinates is activated by adding the keyword `Coordinates` in the parentheses where the domain is specified.

The average, integral, maximum, and minimum can be confined to a window by using the window specification. A 1D, 2D, or 3D window is defined by the coordinates (in micrometers) of two opposite corners of the window. In addition, you can confine the domain in a well using the well specification. In this case, the well is defined by the coordinates of a point inside the well. When a list of domains is specified in addition to the window or well, the intersection of the window with the specified domains is taken into account for the average, integral, maximum, or minimum.

The length unit for integration and the number of digits used for names in the current plot file can be specified in the `Math` section (see [CurrentPlot Options on page 169](#)).

Parameters from the parameter file of Sentaurus Device can also be added to the current plot file. The general specification looks like:

```
[ Material = <material> | MaterialInterface = <interface> |
  Region = <region> | RegionInterface = <interface> ]
  Model = <model> Parameter = <parameter>
```

Specifying the location (material, material interface, region, or region interface) is optional. However, the model name and parameter name must always be present.

**Note:**

The order of specification is fixed. An optional location specification (material, material interface, region, or region interface) must be followed by a model name and a parameter name. Otherwise, Sentaurus Device will report a syntax error.

[Ramping Physical Parameter Values on page 126](#) describes how model names and parameter names can be determined.

Sentaurus Device also provides a current plot PMI (see [Current Plot File on page 1439](#)).

**Note:**

Do not confuse the `CurrentPlot` section with the `CurrentPlot` statement in the `Solve` section introduced in [When to Write to the Current File on page 162](#).

## Chapter 5: Simulation Results

### Current File

## Example: Mixed Mode

In mixed-mode simulations, the `CurrentPlot` section can appear in the body of a physical device within the `System` section (it is also possible to have a global `CurrentPlot` section). For example:

```
System {
    Set (gnd = 0)
    CAP Cm (Top=node2 Bot=gnd) {CurrentPlot {Potential ((0.7, 0.8, 0.9))}}
    ...
}
```

## Example: Advanced Options

This example is a 2D device that uses the more advanced `CurrentPlot` features:

```
CurrentPlot {
    hDensity((0 1)) * hole density at position (0um, 1um)
    ElectricField/Vector((0 1)) * Electric Field Vector
    Potential (
        (0.1 -0.2) * coordinates need not be integers
        Average(Region="Channel") * average over a region
        Average(Everywhere) * average over entire device
        Maximum(Material="Oxide") * Maximum in a material
        Maximum(Semiconductor) * in all semiconductors
        * minimum in a material and a region, output under the name "x":
        Minimum(Name="x" Region="Channel" DopingWell(-0.1 0.3))
    )
    eDensity(
        * average and coordinates of centroid
        Average(Semiconductor Coordinates)
        * maximum and coordinates of maximum in well
        Maximum(DopingWell(-0.1 0.3) Coordinates)
        * integral over the semiconductor regions
        Integrate(Semiconductor)
    )
    SpaceCharge(
        * maximum over 2D window
        Maximum(Window[(-0.2 0) (0.2 0.2)])
        * integral over well
        Integrate( DopingWell(-0.1 0.3) )
    )
}
```

### Note:

If you specify multiple domains in addition to the window or well, the intersection of the window with the specified domains is taken into account for the average, integral, maximum, or minimum.

## Chapter 5: Simulation Results

### Current File

## Example: Plotting Parameter Values

The following example adds five curves to the current plot file:

```
CurrentPlot {
    Model = DeviceTemperature Parameter = "Temperature"
    Material = Silicon Model = Epsilon Parameter = epsilon
    MaterialInterface = "AlGaAs/InGaAs"
    Model = "SurfaceRecombination" Parameter = "S0_e"
    Region = "bulk" Model = LatticeHeatCapacity Parameter = cv
    RegionInterface = "Region.0/Region.1"
    Model = "SurfaceRecombination" Parameter = "S0_h"
}
```

This example also shows the fixed order of specification. An optional location (material, material interface, region, or region interface) is followed by a model name and a parameter name. Therefore this example adds the following five parameter values to the current plot file:

1. Global device temperature
2. Dielectric constant  $\epsilon$  in silicon
3. Surface recombination parameter  $s_0$  for electrons on AlGaAs–InGaAs material interfaces
4. Lattice heat parameter  $cv$  for region ‘bulk’
5. Surface recombination parameter  $s_0$  for holes on the Region.0–Region.1 region interface

## CurrentPlot Options

The length unit used during a current plot integration (see [Tracking Additional Data in the Current File on page 165](#)) can be selected as follows:

```
Math {
    CurrentPlot (IntegrationUnit = um)
}
```

The options are `cm` (centimeters) and `um` (micrometers). The default is `um`. This keyword is useful when quantities such as densities ( $\text{unit of cm}^{-3}$ ) are integrated. In a 2D simulation, the unit of the integral is either  $\mu\text{m}^2\text{cm}^{-3}$  (`IntegrationUnit=um`) or  $\text{cm}^{-1}$  (`IntegrationUnit=cm`).

The number of digits in the names of quantities in the current plot file can be specified as follows:

```
Math {
    CurrentPlot (Digits = 6)
}
```

## Chapter 5: Simulation Results

### Current File

The default is `Digits=6`. This keyword is useful when the names of two quantities become equal due to rounding. In the following example, you want to monitor the valence band energy in two points:

```
CurrentPlot {  
    ValenceBandEnergy ((5.0, 199.011111) (5.0, 199.011112))  
}
```

However, with the default setting of `Digits=6`, both quantities are assigned the same name due to rounding:

```
Pos(5,199.011) ValenceBandEnergy
```

By increasing the number of digits, the names in the current plot file become distinct.

Specifying `Digits` affects the names of the following quantities in the current plot file:

- Coordinates
- Well specification
- Window specification

---

## Tcl Formulas

Sentaurus Device can evaluate Tcl formulas and add the results to the current plot file. The Tcl interpreter has access to the data listed in [Appendix F on page 1515](#), and you can provide Tcl commands to compute derived quantities. For example, it is possible to compute the electron conductivity  $\sigma_n$  given by:

$$\sigma_n = qn\mu_n \quad (30)$$

The following operations can be performed with a Tcl formula:

- Evaluation at a given vertex
- Evaluation at a location specified by its coordinates
- Compute the minimum/maximum/average/integral over a domain
- Plot output to a PMI user field

In the case of `Minimum`, `Maximum`, `Average`, or `Integrate` operations or a plot operation, the domain of the evaluation can be restricted as follows:

- Region
- Material
- Region interface
- Material interface

## Chapter 5: Simulation Results

### Current File

- Contact
- Semiconductor, insulator, or metal regions
- Entire device
- Doping well
- Window

In the case of Minimum, Maximum, Average, or Integrate operations, the location of the minimum or maximum, or the centroid of the average or integral operation (see [Equation 29](#)) also will be added to the current plot file.

The following example shows how to compute the average electron conductivity  $\sigma_n$  in the channel region:

```
CurrentPlot {
    Tcl (
        Dataset      = "Ave_channel eConductivity"
        Function     = "Conductivity"
        Formula      = "set q 1.602e-19
                        set n [tcl_cp_ReadScalar eDensity]
                        set mu [tcl_cp_ReadScalar eMobility]
                        set value [expr $q * $n * $mu]"
        Operation   = "Average Region = channel"
    )
}
```

The `Tcl` statement supports the following options:

```
CurrentPlot {
    Tcl (
        Dataset      = "..."
        Function     = "..."
        Unit         = "..."
        Init          = "..."
        Formula      = "..."
        Finish        = "..."
        Operation   = "..."
    )
}
```

### Dataset Option

This option specifies the dataset name that appears in the header section of the current plot file. If it is not specified, Sentaurus Device generates the name `Tcl_Dataset_<index>`, where `<index>` is a unique integer. For example:

```
Dataset = "channel eConductivity"
```

## Function Option

This is the function name that appears in the header section of the current plot file. If it is not specified, Sentaurus Device generates the name `Tcl_Function_<index>`, where `<index>` is a unique integer. For example:

```
Function = "Conductivity"
```

## Unit Option

This is the unit of the current plot quantity. The `.plt` file format does not support units. For example:

```
Unit = "cm/s"
```

## Init Option

The Tcl code in this option is executed first for each plot time point. It can be used to initialize quantities that will be referenced in `Formula`. For example:

```
Init = "set counter 0"
```

## Formula Option

The Tcl code in this option is executed on individual mesh vertices. It must evaluate a formula and assign the result to the Tcl variable `value`.

The following Tcl functions and variables are available:

- `tcl_cp_dim`: Dimension of the mesh; possible values are 1, 2, or 3
- `tcl_cp_vertex`: Mesh index of the current vertex
- `tcl_cp_ContactOuterVoltage`: Returns the value of the contact voltage for the local contact vertex (the evaluation domain must be contact)
- `tcl_cp_ContactInnerVoltage`: Returns the value of the contact inner voltage for the local contact vertex (the evaluation domain must be contact)
- `proc tcl_cp_ReadScalar {dataname}`: Returns the value of a scalar data field for the local vertex (see [Appendix F on page 1515](#) for valid data names)
- `proc tcl_cp_ReadVector {dataname index}`: Returns a component of a vector data field for the local vertex

The parameter `index` must satisfy `0 <= $index < $tcl_cp_dim`. See [Appendix F on page 1515](#) for valid data names.

## Chapter 5: Simulation Results

### Current File

- `proc tcl_cp_WriteScalar {dataname value}`: This Tcl function defines the value of a scalar field for the local vertex. This is a side effect of the Tcl current plot statement, and it is typically used to compute the values of a PMI user field.

Alternatively, you can use the `Plot` operation (see [Operation Option on page 173](#)).

For example:

```
Formula = "incr counter  
          set value [tcl_cp_ReadScalar ElectrostaticPotential]"
```

## Finish Option

The Tcl code in this option is executed last for each plot time point. It can be used for postprocessing purposes. It can access the Tcl list `result`, which contains the final current plot value, followed by the coordinates. When the operation is `Minimum`, `Maximum`, `Average`, or `Integrate` (see [Operation Option](#)), the coordinates are those of the corresponding extremum or the centroid (see [Equation 29](#)). For example:

```
Finish = "puts \"Used $counter calls\""
```

For additional examples, see [Examples on page 175](#).

## Operation Option

This option determines how the current plot formula is evaluated. The following operations are recognized:

- `Node=<int>`: Evaluate formula at specified node.
- `Coordinate= ( . . . )`: Evaluate formula at specified coordinates.
- `Minimum`: Compute minimum over specified domain.
- `Maximum`: Compute maximum over specified domain.
- `Average`: Compute average over specified domain.
- `Integrate`: Compute integral over specified domain.
- `Plot=<name>`: Output formula to PMI user field.

### Note:

Only one of these operations must be specified.

To obtain the value of the current plot formula at specified coordinates, Sentaurus Device evaluates the formula at nearby vertices and interpolates these values.

## Chapter 5: Simulation Results

### Current File

The required interpolation scheme can be selected by specifying

`Interpolation=<string>`. The available interpolation schemes are `Arsinh`, `Linear` (default), and `Logarithmic`.

A scaling factor for `arsinh` interpolation also can be specified:

- `ArsinhFactor=<value>`: Specify `arsinh` scaling factor. The default is 1.

The length unit when computing an integral can be selected as follows:

- `IntegrationUnit=<unit>`: Specify integration unit.

The options are `cm` (centimeter) and `um` (micrometer). The default is `um`. This keyword is useful when quantities such as densities (unit of  $\text{cm}^{-3}$ ) are integrated. In a 2D simulation, the unit of the integral is either  $\mu\text{m cm}^{-3}$  (`IntegrationUnit=um`) or  $\text{cm}^{-1}$  (`IntegrationUnit=cm`).

In the case of `Minimum`, `Maximum`, `Average`, or `Integrate` operations or a plot operation, the required domain also can be specified as follows:

- `Region=<regionname>`: Region domain
- `Material=<materialname>`: Material domain
- `RegionInterface=<region1/region2>`: Region interface domain
- `MaterialInterface=<material1/material2>`: Material interface domain
- `Contact=<contactname>`: Contact domain (for minimum, maximum, average, or integral operation only)
- `Semiconductor`: Use all semiconductor regions as domain
- `Insulator`: Use all insulator regions as domain
- `Conductor`: Use all metal regions as domain
- `Everywhere`: Use entire device as domain (default)
- `DopingWell= (...)`: Doping well domain
- `Window= [(...)]`: Window domain

If multiple domains are specified, Sentaurus Device will only evaluate the current plot formula where the domains intersect. For example, it is possible to specify:

```
Operation = "Average Region=drain DopingWell=(1 2)"
```

In this case, the formula is only evaluated for those vertices in the drain region that also lie within the specified doping well.

## Chapter 5: Simulation Results

### Current File

Region interface, material interface, and contact specifications cannot be combined with other domains. For example:

```
Operation = "Average Window=[(1 2) (3 4)]"
```

For the contact specification, when the contact is connected to multiple regions, you have the option to select a subset of the contact vertices involved in the operation (the intersection between the contact and a selected region). The selected region is specified by the `Region` keyword.

For example, to integrate a formula over the intersection of contact anode and region `r1`, specify:

```
Operation = "Integrate Contact=\"anode\" Region=\"r1\""
```

## Examples

Evaluate the electrostatic potential  $\phi$  at (7.2  $\mu\text{m}$ , 2.1  $\mu\text{m}$ ):

```
CurrentPlot {
    Tcl (
        Formula = "set value [tcl_cp_ReadScalar ElectrostaticPotential]"
        Operation = "Coordinate = (7.2 2.1)"
        Dataset = "7.2_2.1 ElectrostaticPotential"
        Function = "ElectrostaticPotential"
    )
}
```

Save the electron conductivity  $\sigma_n = qn\mu_n$  in semiconductor (see [Equation 30 on page 170](#)) as a PMI user field:

```
CurrentPlot {
    Tcl (
        Formula = "set q 1.602e-19
                   set n [tcl_cp_ReadScalar eDensity]
                   set mu [tcl_cp_ReadScalar eMobility]
                   set value [expr $q * $n * $mu]"
        Unit = "Ohm^-1*cm^-1"
        Operation = "Plot = PMIUserField5 Semiconductor"
    )
}
```

Plot the integral of the space charge  $\rho$  in the window [(0  $\mu\text{m}$ , 6  $\mu\text{m}$ ) (2.5  $\mu\text{m}$ , 7  $\mu\text{m}$ )]:

```
CurrentPlot {
    Tcl (
        Formula = "set value [tcl_cp_ReadScalar SpaceCharge]"
        Operation = "Integrate Window = [(0 6) (2.5 7)] IntegrationUnit = cm"
        Dataset = "Integral_Window SpaceCharge"
        Function = "SpaceCharge"
    )
}
```

## Chapter 5: Simulation Results

### Current File

→ →  
Compute the integral of the electron Joule heat  $J_nE$  in semiconductor regions:

```
CurrentPlot {
    Tcl (
        Formula = "set value 0
                    for {set d 0} {$d < $tcl_cp_dim} {incr d} {
                        set j [tcl_cp_ReadVector eCurrentDensity $d]
                        set f [tcl_cp_ReadVector ElectricField $d]
                        set value [expr $value + $j * $f]
                    }"
        Operation = "Integrate Semiconductor IntegrationUnit = cm"
        Dataset = "Integral_Semiconductor eJouleHeat"
        Function = "JouleHeat"
    )
}
```

Compute the distributed conductance of the contact anode:

```
CurrentPlot {
    Tcl (
        Formula = "set value 0
                    set Vext [tcl_cp_ContactOuterVoltage \"anode\"]
                    set QF [tcl_cp_ReadScalar \"QuasiFermiPotential\"]
                    for {set d 0} {$d < $tcl_cp_dim} {incr d} {
                        set j [tcl_cp_ReadVector eCurrentDensity $d]
                        set N [tcl_cp_ReadVector ContactSurfaceNormal $d]
                        set value [expr $value + $j * $N]
                    }
                    set value [expr $value / ($Vext - $QF + 0.0001)]"
        Operation = "Integrate Contact=\"top1\" Region=\"r2\""
                    IntegrationUnit = cm"
        Dataset = "Conductance"
        Function = "Conductance"
    )
}
```

Compute an integral value and use the `Finish` statement. To compute the integral of the optical generation inside silicon and to write the result to the log file, use the `Finish` option as follows:

```
CurrentPlot {
    Tcl (
        Formula = "set value [tcl_cp_ReadScalar OpticalGeneration]"
        Operation = "Integrate Material=Silicon"
        Dataset = "IntegrSilicon G"
        Function = "OpticalGeneration"
        Finish =
            set v [lindex $result 0]
            set x [lindex $result 1]
            set y [lindex $result 2]
            puts [format \"GSi = %.4g; centroid_coord=(%.4g %.4g)\" $v $x $y]
    )
}
```

## Chapter 5: Simulation Results

### Device Plots

In this case, the log file will contain the following information:

```
GSi = 1.669e+21; centroid_coord=(1.718 0.5995)
```

---

## Device Plots

Device plots show spatial-dependent datasets in the device and provide a view of the inside of the device.

---

### What to Plot

The `Plot` section specifies the data that is saved at the end of or, optionally, during the simulation to the `Plot` file specified in the `File` section or by the `Plot` command in the `Solve` section. Consecutive plots can be collected in a single file, or written to separate enumerated files, or written to a single file by overwriting the previous plot. To collect consecutive plots in a single file, specify `Plot(collected)=<filename>` in the `File` section.

The keywords `Overwrite` and `noOverwrite`, specified as options in the `Plot` command in the `Solve` section (see [When to Plot on page 178](#)), control whether plots are written to separate files or to a single file.

**Note:**

You can control further compression, beyond a basic standard compression of `Plot` files, by specifying `CompressTDR` in the `Math` section. However, further compression might not always lead to a significantly reduced file size and, depending on the content of the file, it might lead to a slightly increased file size.

Specifying `CompressTDR` in the `Math` section applies to all types of plot commands that generate files containing spatial-dependent datasets.

Vector data can be plotted by appending `/Vector` to the corresponding keyword. For example:

```
Plot { ElectricField/Vector }
```

Element-based scalar data can be plotted by appending `/Element` to the corresponding keyword. For example:

```
Plot { eMobility/Element }
```

Some quantities (such as the carrier densities) are put into the `Plot` file even when they are not listed in the `Plot` section. The keyword `PlotExplicit` in the global `Math` section suppresses this behavior.

## Chapter 5: Simulation Results

### Device Plots

By default, the plot file also contains additional information required by `Load` to restart a simulation (see [Save and Load Statements on page 213](#)). To suppress writing this additional information, specify `-PlotLoadable` in the global `Math` section.

You can specify the keyword `DatasetsFromGrid` to copy quantities from the TDR grid file directly to the plot file. You can copy all datasets by specifying `DatasetsFromGrid` by itself:

```
Plot { DatasetsFromGrid }
```

Alternatively, you can copy selected quantities by listing their names:

```
Plot {  
    DatasetsFromGrid (dataset1 dataset2 dataset3)  
}
```

A dataset is only copied from the TDR grid file if it is not computed by Sentaurus Device. Otherwise, the dataset computed by Sentaurus Device takes precedence.

**Note:**

The keyword `DatasetsFromGrid` only copies entire datasets from the TDR grid file to the plot file. You cannot copy partial datasets, for example, the components of tensor and vector datasets.

See [Table 334 on page 1713](#) for all possible plot options.

---

## When to Plot

The simplest way to create a device plot is to define `Plot` in the `File` section. By default, the output to the `Plot` file will occur at the end of the simulation only.

The commands for `Quasistationary` and `Transient` analysis support an option `Plot` that allows you to write plots while these commands are executing. The plot file name is derived from the one specified with `Plot` in the `File` section. See [Saving and Plotting During a Quasistationary on page 130](#) and [Transient Command on page 138](#) for the syntax.

The most flexible way to create device plots is through the `Plot` statement in the `Solve` section. It provides full control of when to plot and limited control of the file names. The `Plot` statement can be used at any level of the `Solve` section. The command takes the form:

```
Plot (<parameters-opt>) <system-opt>
```

If `<system-opt>` is not specified, then all physical devices and circuits are plotted. Use `<system-opt>` to specify an optional list of devices delimited by braces (see [Table 337 on page 1715](#)).

If `<parameters-opt>` is not specified, then defaults are used. The keywords `Loadable` and `Explicit` provide plot-specific overrides for `PlotLoadable` and `PlotExplicit` in the global

## Chapter 5: Simulation Results

### Device Plots

Math section (see [What to Plot on page 177](#)). For a summary of all options, see [Table 345 on page 1727](#).

#### Example

```
Solve {
    Plugin {
        Poisson
        Plot ( FilePrefix = "output/poisson" )
        Coupled { Poisson Electron Hole }
        Plot (FilePrefix = "output/electric" noOverwrite)
    }
    Transient {
        Coupled { Poisson Electron Hole Temperature }
        Plot ( FilePrefix = "output/trans"
            Time = ( range = (0 1) ;
                      range = (0 1) intervals = 4 ; 0.7 ;
                      range = (1.e-3 1.e-1) intervals = 2 decade )
            NoOverwrite )
    }
    ...
}
```

The first `Plot` statement in `Plugin` writes (after the computation of the Poisson equation) to a file named `output/poisson_des.tdr`. The second `Plot` statement in `Plugin` writes to a file called `output/electric_0000_des.tdr` and increases the internal number for each call.

The `Plot` statement in the transient specifies three different types of time entries (separated by semicolons). The first entry indicates all the times within this range when a plot file must be written. The second time entry forces the transient simulation to compute solutions for the given times. In this example, the given times are 0.25, 0.5, 0.75, and 1.0. The third entry is for the single time of 0.7. The fourth entry triggers the plotting at time points given by subdividing the range on the logarithmic scale, resulting for this example in plots at 1.e-3, 1.e-2, and 1.e-1.

---

## Snapshots

Sentaurus Device offers the possibility to save snapshots interactively during a simulation. This can be undertaken by sending a POSIX signal to the Sentaurus Device process.

Depending on whether `Plot` or `Save` or both is specified in the `File` section, the signal invokes a request to write a plot (`.tdr`) file or a save (`.sav`) file after the actual time step is finished.

## Chapter 5: Simulation Results

### Device Plots

The following signals are supported to save snapshots:

- USR1: The occurrence of the USR1 signal initiates Sentaurus Device to write a plot file or a save file after the simulation has finished its actual time step. The simulation will continue afterwards.
- INT: By default, sending the INT signal causes a process to exit. This behaviour changes if `Interrupt=BreakRequest` or `Interrupt=PlotRequest` is specified in the `Math` section (see [Table 211 on page 1573](#)). In both cases, the occurrence of the INT signal initiates Sentaurus Device to write a plot file or a save file. `Interrupt=BreakRequest` causes Sentaurus Device to terminate the actual solve statement after the snapshot is saved. If `Interrupt=PlotRequest` is specified, the simulation continues.

A signal can be sent to a process by invoking the `kill` command (see the corresponding man page of your operating system). The process ID can be extracted from the `.log` file.

---

## Interface Plots

Data fields defined on interfaces can be plotted by using the modifier `/RegionInterface`. For example:

```
Plot {  
    eTrappedCharge/RegionInterface  
}
```

The following fields are available for interface plots:

```
TotalTrapConcentration  
eTrappedCharge  
hTrappedCharge  
eHIDensity  
hHIDensity  
eHIDensityPerDensity  
hHIDensityPerDensity  
eHIDensityPerTemperature  
hHIDensityPerTemperature  
eHIDensityPer_eTemperature  
hHIDensityPer_hTemperature
```

Interface plots are generated only if interface regions appear in the grid file.

## Plotting Results Along Carrier Paths

This section discusses how to plot results along carrier paths.

### Saddle Point Analysis in CIS Devices

A design parameter of CMOS image sensor (CIS) devices is the energy barrier for carriers formed in the transition between the photodiode and the gate channel. This transition region corresponds to a saddle point in the landscape shaped by the electrostatic potential.

Sentaurus Device can perform saddle point analysis in CIS devices. Using its path search capability, Sentaurus Device follows the majority current from a set of starting points. The paths are saved as 1D datasets in a separate TDR file that can be overlaid with the device structure TDR file. The path data can be plotted (for example, path length versus band energy) in xy plot mode using Sentaurus Visual.

For every defined starting point, Sentaurus Device follows the majority carrier current lines:

- If the local net doping is n-type,  $n > p$ , then it follows the negative majority electron current vector  $\vec{-J_e}$ .
- If the local net doping is p-type,  $p \geq n$ , then it follows the majority hole current vector  $\vec{J_h}$ .

The `ComputeCarrierPath` section in the `Math` section controls the computation of carrier paths for saddle point analysis.

#### Start Condition

The starting points of the path search are defined in the `Start` statement. Different shapes can be specified to define the starting points for the path search: doping well, cuboid, and line. There are two possibilities to define the starting points within a defined shape:

- Start from vertex positions located inside the shape
- Start from a tensor subgrid located inside the shape

#### Start From Vertex Positions: Doping Well

Typically, the starting points are located inside a specific doping well. You specify a coordinate inside the doping well by using `DopingWell(point=<vector>)`, and Sentaurus Device detects the corresponding doping well. All vertices that are located inside the doping well are used as starting points for the path search. For example:

```
ComputeCarrierPath(  
    Start(  
        DopingWell( point=(4.23 3.88 12.6) )  
    )  
)
```

## Chapter 5: Simulation Results

### Device Plots

#### Start From Vertex Positions: Cuboid

You specify a cuboid by using `Cuboid(corner1=(x1 y1 z1) corner2=(x2 y2 z2))`, where `corner1` and `corner2` are the diagonally opposite corners of the cuboid. All vertices that are located inside the cuboid are used as starting points for the path search. For example:

```
ComputeCarrierPath(  
    Start(  
        Cuboid( corner1=(0.92 2.3 0.3) corner2=(3.0 3.5 3.4) )  
    )  
)
```

#### Start From Vertex Positions: Line

You specify a line by using `Line(point1=(x1 y1 z1) point2=(x2 y2 z2))`, where `point1` and `point2` mark the start and end of the line, respectively. All vertices that are located on the line are used as starting points for the path search. For example:

```
ComputeCarrierPath(  
    Start(  
        Line( point1=(1 0 0) point2=(2 0 0) )  
    )  
)
```

#### Start From Tensor Subgrid

In the `DopingWell`, `Cuboid`, and `Line` statements, you can specify the tensor subgrid by using `grid=<vector>`. Sentaurus Device constructs the equidistant subgrid around the point of origin. The point of origin is defined as `point` for `DopingWell`, `corner1` for `Cuboid`, and `point1` for `Line`. For example:

```
ComputeCarrierPath(  
    Start(  
        DopingWell( point=(4.23 3.88 12.6) grid=(0.1 0.2 0.3) )  
        Cuboid( corner1=(0.9 2.3 0.3) corner2=(3.0 3.5 3.4)  
            grid=(0.2 0.1 0.5) )  
        Line( point1=(1 0 0) point2=(2 0 0) grid=(0.05) )  
    )  
)
```

#### End Condition

By default, the path search stops when it reaches a contact. Typically, it is sufficient to stop the path search when it reaches the end doping well. The end doping well can be specified using the `DopingWell` statement in the `End` statement. For example:

```
ComputeCarrierPath(  
    End(  
        DopingWell( point=(1.23 0.88 3.6) )  
    )  
)
```

## Chapter 5: Simulation Results

### Device Plots

#### Saving Carrier Paths as 1D Datasets in Separate TDR File

You can specify the file name of a separate output TDR file in the `File` section using the keyword `CarrierPath`. For example:

```
File{
    CarrierPath = "myFileName.tdr"
}
```

#### Triggering the Path Search

The computation of carrier paths is triggered when TDR files are being saved. You can have better control of the computation of carrier paths by using the syntax from the `Plot` section inside the `Transient` statement. For example:

```
Transient(
    Initialtime= 0 Finaltime= 4.0e-9
    Initialstep= 1e-9 Minstep= 1e-12 Maxstep= 1.2e-06
    Increment= 1.5 Decrement= 3.0
){
    Coupled {Poisson Electron Hole}
    Plot (FilePrefix="myPrefix" time=(1.0e-9; 3.0e-9) nooverwrite)
}
```

---

## Log File

The name of the log file is specified by `output` in the `File` section. The log file contains a copy of the messages that Sentaurus Device prints while a simulation runs. The log file contains various information, including:

- Technical information such as the version of Sentaurus Device that is running, the machine where it is running, and the process ID
- Confirmation as to which structure is simulated, which physical models have been selected, and which parameters are used
- Detailed information about how the simulation proceeds, including timing and convergence information
- Warning messages

The log file is of minor interest as long as your simulations are set up well and no convergence problems occur. It is a valuable diagnostic tool during simulation setup or when convergence problems occur.

A log file annotated with XML tags can be generated by specifying the `--xml` command-line option. This file contains the same information as the regular log file, but features additional XML tags to organize its content. The XML log file uses the same file name as the regular log file, but with the extension `.xml` instead of `.log`.

## Chapter 5: Simulation Results

### Extraction File

The XML log file is best displayed with the TCAD Log File Browser. For more information, see *Utilities User Guide*, Chapter 2.

---

## Extraction File

Sentaurus Device supports a special-purpose file format (extension .xtr) for the extraction of MOSFET compact model parameters.

---

### Extraction File Format

The extraction file consists of the following parts:

- A header identifies the file format.
- A section contains the process information of the MOSFET such as channel length or channel width.
- A collection of curves contains the values of a dependent variable as a function of an independent variable, for example, drain current versus gate voltage in an  $I_d$ - $V_g$  ramp. In addition, each curve contains the corresponding bias conditions, for example,  $V_b$ ,  $V_d$ , and  $V_s$  in an  $I_d$ - $V_g$  ramp.

A sample extraction file is:

```
# Synopsys Extraction File Format Version 1.1
# Copyright (C) 2008 Synopsys, Inc.
# All rights reserved.

$ Process
AD 31.50f
AS 31.50f
D NMOS
L 90.00n
NF 1.000
NRD 0.000
NRS 0.000
PD 880.0n
PS 880.0n
SA 500.0n
SB 500.0n
SC 0.000
SCA 0.000
SCB 0.000
SCC 0.000
SD 0.000
W 90.00n

$ Data
```

## Chapter 5: Simulation Results

### Extraction File

```

Curve: Id_Vg
Bias : Vb = 0 , Vd = 0.5 , Vs = 0
1.60000000000000E+00    7.95091490486384E-05
1.63750000000000E+00    8.27433425335473E-05
1.67500000000000E+00    8.59289596870642E-05
1.71250000000000E+00    8.90665609661286E-05
1.75000000000000E+00    9.21567959785304E-05

Curve: Is_Vg
Bias : Vb = 0 , Vd = 0.5 , Vs = 0
1.60000000000000E+00    -7.95091490484525E-05
1.63750000000000E+00    -8.27433425333610E-05
1.67500000000000E+00    -8.59289596868774E-05
1.71250000000000E+00    -8.90665609659414E-05
1.75000000000000E+00    -9.21567959783428E-05

```

## Analysis Modes

Sentaurus Device supports different analysis modes.

*Table 20 Analysis modes*

Analysis	Curve <sup>1</sup>	Description
DC	II_i_Vj	The $i$ -th terminal current versus $j$ -th terminal voltage, where $i,j = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: all terminal voltages other than $j$ -th terminal voltage
AC	Ai_j_Vk	Admittance $A_{ij}$ versus $k$ -th terminal voltage, where $i,j,k = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: all terminal voltages other than $k$ -th terminal voltage and frequency
	Ai_j_F	Admittance $A_{ij}$ versus frequency, where $i,j = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: $V_b, V_d, V_g, V_s$
	Ci_j_Vk	Capacitance $C_{ij}$ versus $k$ -th terminal voltage, where $i,j,k = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: all terminal voltages other than $k$ -th terminal voltage and frequency
	Ci_j_F	Capacitance $C_{ij}$ versus frequency, where $i,j = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: $V_b, V_d, V_g, V_s$

## Chapter 5: Simulation Results

### Extraction File

Table 20 Analysis modes (Continued)

Analysis	Curve <sup>1</sup>	Description
Noise	noise_Vi_Vj	Autocorrelation noise voltage spectral density (NVSD) for electrode $i$ versus $j$ -th terminal voltage, where $i,j = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: all terminal voltages other than $j$ -th terminal voltage and frequency
	noise_Vi_F	Autocorrelation noise voltage spectral density (NVSD) for electrode $i$ versus frequency, where $i = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: $V_b, V_d, V_g, V_s$
	noise_Ii_Vj	Autocorrelation noise current spectral density (NISD) for electrode $i$ versus $j$ -th terminal voltage, where $i,j = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: all terminal voltages other than $j$ -th terminal voltage and frequency
	noise_Ii_F	Autocorrelation noise current spectral density (NISD) for electrode $i$ versus frequency, where $i = b$ (bulk), $d$ (drain), $g$ (gate), or $s$ (source). Bias conditions: $V_b, V_d, V_g, V_s$

1. Underscores are optional and are used to improve readability.

---

## File Section

You specify the name of the extraction file in the `File` section of the Sentaurus Device command file. For example:

```
File {
    Extraction = "mosfet_des.xtr"
    ...
}
```

The default file name is `extraction_des.xtr`.

---

## Electrode Section

Sentaurus Device automatically recognizes the four electrodes of a MOSFET if they are called "bulk", "drain", "gate", and "source", or "b", "d", "g", and "s" (case insensitive).

## Chapter 5: Simulation Results

### Extraction File

For other contact names, you can specify the mapping in the `Electrode` section:

```
Electrode {
  { Name = "contact_bulk"  Voltage = 0.0 Extraction {bulk}  }
  { Name = "contact_drain" Voltage = 0.0 Extraction {drain} }
  { Name = "contact_gate"  Voltage = 0.0 Extraction {gate}  }
  { Name = "contact_source" Voltage = 0.0 Extraction {source} }
}
```

All four MOSFET electrodes (bulk, drain, gate, and source) must be present. Additional electrodes are allowed, but they will be ignored for extraction purposes.

---

## Extraction Section

The process parameters can be part of the Sentaurus Device grid or doping file. You can also specify the same information in an `Extraction` section. For example:

```
Extraction {
  AD 31.50f
  AS 31.50f
  D NMOS
  L 90.00n
  NF 1.000
  NRD 0.000
  NRS 0.000
  PD 880.0n
  PS 880.0n
  SA 500.0n
  SB 500.0n
  SC 0.000
  SCA 0.000
  SCB 0.000
  SCC 0.000
  SD 0.000
  W 90.00n
}
```

Each line in the `Extraction` section consists of a name–value pair. All entries are copied to the extraction file as is.

#### Note:

The process parameters in the `Extraction` section of the Sentaurus Device command file take precedence over the information in the grid or doping file.

---

## Solve Section

The `Quasistationary` statement in the `Solve` section supports an `Extraction` option to request voltage-dependent extraction curves. Multiple curves can be generated during a

## Chapter 5: Simulation Results

### Extraction File

single ramp. No extraction curves are produced if the Extraction option is missing. For example:

```
Solve {
    # ramp contributes to extraction
    Quasistationary (
        Goal { Name="contact_gate" Voltage=1.5 }
        Extraction { IdVg IsVg ... }
    )
    { Coupled { Poisson Electron }
        CurrentPlot (Time = (Range=(0 1) Intervals=10))
    }

    # ramp does not contribute to extraction
    Quasistationary (
        Goal { Name="contact_base" Voltage=0.5 }
    )
    { Coupled { Poisson Electron } }

    # ramp contributes to extraction
    Quasistationary (
        Goal { Name="contact_drain" Voltage=1.5 }
        Extraction { IdVd IbVd ... }
    )
    { Coupled { Poisson Electron } }
}
```

AC and noise simulations must be performed in mixed mode. Only one physical device is supported in the circuit. Voltage-dependent curves are specified as an option to the Quasistationary statement; whereas, frequency-dependent curves appear within the ACCoupled statement:

```
Solve {
    Quasistationary (
        ...
        Extraction { Ads_Vg Agg_Vg Cgd_Vg noise_Id_Vg noise_Ig_Vg }
    )
    { ACCoupled (
        ...
        Extraction { Add_F Cgd_F Csd_F
                    noise_Vd_F noise_Vg_F noise_Is_F }
    )
        { Poisson Electron }
    }
}
```

The recognized curves in the Extraction option are shown in [Table 20 on page 185](#).

# 6

## Numeric and Software-Related Issues

---

*This chapter discusses technical issues of simulation that do not have real-world equivalents.*

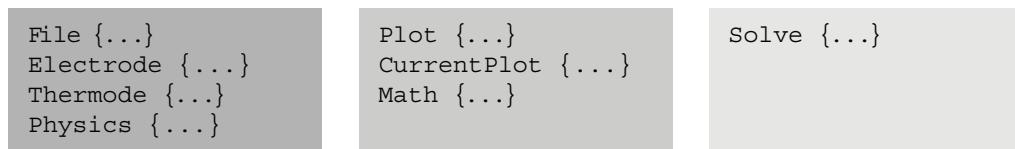
To successfully perform an experiment, apart from understanding the device under investigation, the experimenter must understand, control, and skillfully use the measurement equipment. The same holds for numeric *experiments*. This chapter describes the *equipment* available in Sentaurus Device for numeric experiments: linear and nonlinear solvers, control of accuracy, convergence monitors, and other software-related facilities. Additional background information on numerics is available in [Chapter 38 on page 1177](#).

---

### Structure of Command File

The Sentaurus Device command file is divided into sections that are defined by a keyword and braces (see [Figure 13](#)). A device is defined by the `File`, `Electrode`, `Thermode`, and `Physics` sections. The solve methods are defined by the `Math` and `Solve` sections. Two sections are used in mixed-mode circuit and device simulation, see [Chapter 3 on page 90](#). This section concentrates on the input to define single-device simulations.

*Figure 13     Different sections of a Sentaurus Device command file*



---

### Inserting Files

An `Insert` directive is available in the command file of Sentaurus Device to incorporate other files:

```
Insert = "filename"
```

## Chapter 6: Numeric and Software-Related Issues

### Solve Section: How the Simulation Proceeds

This directive can appear at the top level in the command file or inside any of the following sections:

- CurrentPlot
- Device
- Electrode
- File
- Math
- MonteCarlo
- NoisePlot
- NonlocalPlot
- Physics
- Plot
- RayTraceBC
- Solve
- System
- Thermo

The following search strategy is used to locate a file:

- The current directory is checked first (highest priority).
- If the environment variable `SDEVICE` exists, then the directory associated with the variable is checked (medium priority).
- The `$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB` directory is checked (lowest priority).

**Note:**

The `insert` directive is also available for parameter files (see [Physical Model Parameters on page 70](#)).

---

## Solve Section: How the Simulation Proceeds

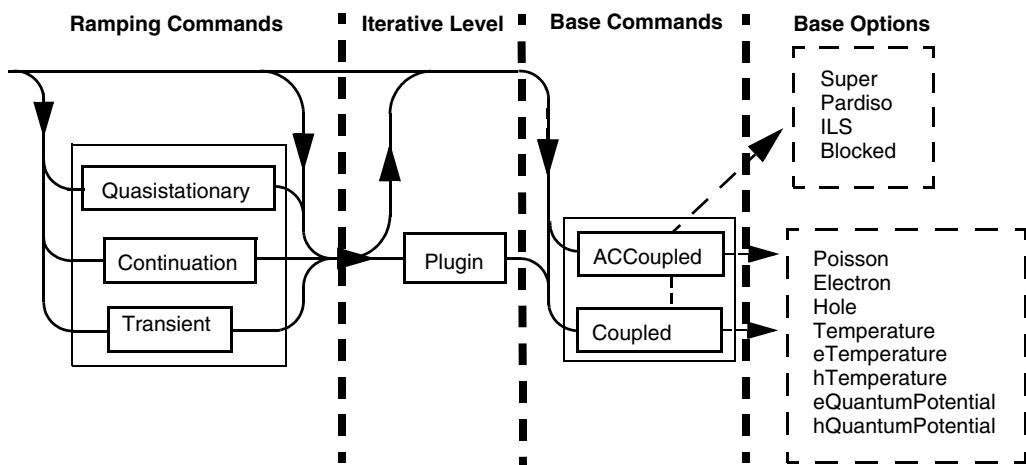
The `Solve` section is the only section in which the order of commands are important. It consists of a series of simulation commands to be performed that are activated sequentially, according to the order of commands in the command file.

Many `Solve` commands are high-level commands that have lower level commands as parameters. [Figure 14](#) shows an example of these different command levels:

- The `Coupled` statement (the base command) is used to solve a set of equations.
- The `Plugin` statement is used to iterate between a number of coupled equations.
- The `Quasistationary` statement is used to ramp a solution from one boundary condition to another.
- The `Transient` statement is used to run a transient simulation.

Furthermore, small-signal AC analysis can be performed with the `ACCoupled` statement. An advanced ramping by continuation method can be performed with the `Continuation` statement. (The `ACCoupled` and `Continuation` statements are presented in [Small-Signal AC Analysis on page 149](#) and [Continuation Command on page 135](#), respectively.)

*Figure 14      Different levels of Solve statements*




---

## Nonlinear Iterations

This section discusses nonlinear iterations.

---

## Coupled Statement

This statement activates a Newton-like solver over a set of equations. Available equations include the Poisson equation, continuity equations, and the different thermal and energy equations. The syntax of the `Coupled` statement is:

```
Coupled ( <optional parameters> ){ <equation> }
```

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

or:

```
<equation>
```

This last form uses only the keyword `equation`, which is equivalent to a coupled with default parameters and the single equation.

For example, if the following command is used, the electrostatic potential and electron density are computed from the resolution of the Poisson equation and electron continuity equation (using the default parameters):

```
Coupled {Poisson Electron}
```

If the following command is used, only the electrostatic potential is computed using the Poisson equation:

```
Poisson
```

The `Coupled` statement is based on a Newton solver. This is an iterative algorithm in which a linear system is solved at each simulation step. Parameters of the command determine:

- The maximum number of iterations allowed
- The required precision of the solution
- The linear solver that must be used
- Whether the solution is allowed to worsen over a number of iterations

These parameters and others are summarized in [Table 340 on page 1719](#).

The following example limits the previous `Coupled {Poisson Electron}` example to ten iterations and uses the ILS linear solver:

```
Coupled ( Iterations=10 Method=ILS ) {Poisson Electron}
```

#### Note:

Use a large number of iterations when the coupled iteration is not inside a ramping process. This allows the Newton algorithm to proceed as far as possible. Inside a ramping statement (for example, `Transient` or `Quasistationary`), the maximum number of iterations should be limited to approximately ten because, if the Newton process does not converge rapidly, it is preferable to try again with a smaller step size than to pursue an iterative process that is not likely to converge.

The best linear method to use in a coupled iteration depends on the type and size of the problem solved. [Table 223 on page 1610](#) lists the solvers and the size of the problems for which they are designed.

## Convergence and Error Control

The `Coupled` statement is sensitive to the following `Math` parameters that determine when a coupled solution diverges or converges:

- `Iterations`
- `RhsAndUpdateConvergence`
- `RhsFactor`, `RhsMin`, `RhsMinFactor`, `RhsMax`, `RhsMaxQ`
- `CheckRhsAfterUpdate`
- `RelErrControl`
- `Digits`
- `ErrRef`
- `Error`
- `UpdateIncrease`, `UpdateMax`

The parameter `Iterations` in the global `Math` section sets the default for `Iterations` of the `Coupled` statement. `Iterations` limits the number of Newton iterations. If the equation being solved is converging quadratically, the number of iterations might increase beyond this limit. For `Iterations=0`, only one iteration is performed.

Convergence of a solution in Sentaurus Device is determined by calculating and examining the following quantities at each Newton iteration:

- *RHS norm* is the norm of the right-hand-side (that is, the residual of the equations).
- *Update error* is a measure of the updates to the equation variables.

These quantities are printed in the log file for each Newton iteration during a solution. By default, a solution is considered converged if the RHS norm < `RhsMin` or if the update error < 1. If `RhsAndUpdateConvergence` is specified, then both of these criteria must be satisfied. However, a solution is accepted as converged if the RHS norm < `RhsMinFactor` × `RhsMin`, even if the update error criterion is not satisfied.

During a single Newton step, the solution is considered diverged if the RHS norm increases by more than a factor of `RhsFactor`. The solution is also considered diverged if the RHS norm exceeds `RhsMax` (for transient simulations) or `RhsMaxQ` (for nontransient simulations).

Specify `CheckRhsAfterUpdate` to force Sentaurus Device to perform an additional check on the RHS norm after the update error criteria is satisfied. If Sentaurus Device assesses that the RHS norm can be made smaller, additional iterations will be performed. This can sometimes improve the quality of a solution and might result in better overall convergence.

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

#### Note:

`CheckRhsAfterUpdate` usually adds no more than one or two extra iterations. However, when convergence is slow or when using extended-precision accuracy, this option can result in several additional iterations if the RHS norm continues to be reduced. In such cases, a larger value for `Iterations` might be needed. In all cases, however, Sentaurus Device will accept a converged solution when the maximum number of iterations is reached, even if the RHS norm is still improving.

There are two formulations of the update error calculations used in Sentaurus Device, depending on whether `RelErrControl` (the default) or `-RelErrControl` is specified. For both formulations, Sentaurus Device tries to determine the value of an equation variable  $x$ , such that the computed update  $\Delta x$  is small enough to satisfy an error criterion:

$$\begin{aligned} \frac{|\Delta x|}{|x| + x_{\text{ref}}} &< \varepsilon_R \quad , \text{ RelErrControl} \\ \frac{|\Delta x/x^*|}{\varepsilon_R |x/x^*| + \varepsilon_A} &< 1 \quad , \text{ -RelErrControl} \end{aligned} \quad (31)$$

where:

- $\varepsilon_R = 10^{-\text{Digits}}$  is the relative error.
- $x_{\text{ref}} = \text{ErrRef}(\text{<equation>})$  is the reference error parameter.
- $\varepsilon_A = \text{Error}(\text{<equation>})$  is the absolute error parameter.
- $x^*$  is a scaling constant for the equation variable.

For large values of  $x$  ( $|x| \rightarrow \infty$ ), the conditions in [Equation 31](#) become relative error criteria:

$$\frac{|\Delta x|}{|x|} < \varepsilon_R \quad (32)$$

Conversely, for small values of  $x$  ( $|x| \rightarrow 0$ ), they become absolute error criteria:

$$|\Delta x| < x_{\text{ref}} \cdot \varepsilon_R \quad \text{or} \quad \frac{|\Delta x|}{x^*} < \varepsilon_A \quad (33)$$

Therefore, [Equation 31](#) ensures a smooth transition between absolute and relative error control.

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

To account for all coupled equations  $i$  at all vertices  $j$ , Sentaurus Device extends the conditions in [Equation 31](#) and defines the update error as:

$$\text{update error} = \sqrt{\frac{\sum_{i=1}^{N_i} \sum_{j=1}^{N_{vi}} \frac{|\Delta x_{i,j}|^2}{\varepsilon_R (|x_{i,j}| + x_{ref,i})}}{N_{vi}}} / \text{RelErrControl} \quad (34)$$

$$\sqrt{\frac{\sum_{i=1}^{N_i} \sum_{j=1}^{N_{vi}} \frac{|\Delta x_{i,j}/x_i^*|^2}{\varepsilon_R |x_{i,j}/x_i^*| + \varepsilon_{A,i}}}{N_{vi}}} / \text{RelErrControl}$$

where:

- $N_i$  is the number of coupled equations.
- $N_{vi}$  is the number of vertices being solved for equation  $i$ .

The value of  $\varepsilon_R$  is defined by specifying the keyword `Digits`.

The values of  $x_{ref}$  and  $\varepsilon_A$  can be defined commonly or independently for each equation with the keywords `ErrRef` and `Error`, respectively. Default values are listed in [Table 342 on page 1722](#).

The parameters `UpdateMax` and `UpdateIncrease` detect divergence of the Newton algorithm. If the update error is larger than `UpdateMax`, or its increase from Newton step to Newton step exceeds `UpdateIncrease`, then the Newton is regarded as diverged.

## Damped Newton Iterations

Line search damping and Bank–Rose damping are two different damping methods. These approaches try to achieve convergence of the coupled iteration far from the final solution by changing the solution by smaller amounts than a normal Newton iteration would. Damping is useful to find an initial solution, but during `Quasistationary` or `Transient` ramps, using smaller time steps is usually preferable.

Line search damping is switched off by default. It is activated when the value of `LineSearchDamping` is set to a value smaller than one. This value determines the minimum factor by which the normal Newton update  $\Delta x$  can be damped. The damping method determines the actual damping factor automatically in each Newton step. If the actual factor falls below the minimum specified by `LineSearchDamping`, line search damping is considered to fail and is disabled for the following Newton iterations.

Bank–Rose damping is a method that automatically adjusts the size of the update based on how the RHS changes [1]. It can be used sometimes to improve convergence when large bias steps are taken.

Bank–Rose damping becomes activated when the number of iterations exceeds the value specified with `NotDamped` in the `Math` section (default is 1000).

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

`LineSearchDamping` and `NotDamped` are set in the global `Math` section and by parameters of the `Coupled` statement. The former specification sets the defaults for the latter.

## Derivatives

For most problems, Newton iterations converge best with full derivatives. Furthermore, for small-signal analysis, and noise and fluctuation analysis, using full derivatives is mandatory. Therefore, by default, Sentaurus Device takes full derivatives into account. For rare occasions where omission of derivatives improves convergence or performance significantly, use the keywords `-AvalDerivatives` and `-Derivatives` in the global `Math` section to switch off mobility and avalanche derivatives.

The derivatives are usually computed analytically, but a numeric computation can be used by specifying `Numerically`. This does not work with the method `Blocked` and, generally, is discouraged.

## Incomplete Newton Algorithm

Certain simple simulations, such as  $I_d$ – $V_g$  ramps, can be accelerated by using a modified Newton algorithm (see [Fully Coupled Solution on page 1203](#) for a description of the standard Newton algorithm). The incomplete Newton algorithm, also known as the Newton–Richardson method, tries to reuse the Jacobian matrix to avoid the expense of evaluating derivatives and computing matrix factorizations.

It can be switched on either in the global `Math` section:

```
Math {  
    IncompleteNewton  
    ...  
}
```

or it can be used as an option in a `Coupled` statement:

```
Coupled (IncompleteNewton) {poisson electron hole}
```

The Jacobian matrix is reused if the following two conditions are satisfied:

$$\|\text{Rhs}_k\| < \text{RhsFactor} \cdot \|\text{Rhs}_{k-1}\| \quad (35)$$

$$\|\text{Update}_k\| < \text{UpdateFactor} \cdot \|\text{Update}_{k-1}\| \quad (36)$$

where  $k$  denotes the iteration index, `Rhs` denotes the right-hand side of the nonlinear equations, and `Update` denotes the Newton step. The values of  $\|\text{Rhs}_k\|$  and  $\|\text{Update}_k\|$  are displayed in the log file of Sentaurus Device during Newton iterations in the columns `|Rhs|` and `|step|`.

You can specify the parameters `RhsFactor` and `UpdateFactor` as arguments to the keyword `IncompleteNewton`. For example:

```
IncompleteNewton (UpdateFactor=0.1 RhsFactor=10)
```

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

The default values are `UpdateFactor=0.1` and `RhsFactor=10`.

## Additional Equations Available in Mixed Mode

The `Contact`, `Circuit`, `TContact`, and `TCircuit` equations are introduced for mixed-mode problems:

- The keyword `Circuit` activates the solution of the electrical circuit models and nodes.
- The keyword `Contact` activates the solution of the electrical interface condition at the contacts.
- Similarly, `TContact` and `TCircuit` activate the solution of the thermal interface condition and the thermal circuit, respectively.

By default, the keywords `Contact` and `Circuit` are not required because the Poisson equation also covers the contact and circuit domains. If only the Poisson equation is solved, no additional equations are added. Only if additional equations to `Poisson` appear in a `Coupled` statement, the circuit and contact equations are also added. Therefore, the following two statements are equivalent:

```
Coupled { Poisson Electron Hole }
Coupled { Poisson Electron Hole Circuit Contact }
```

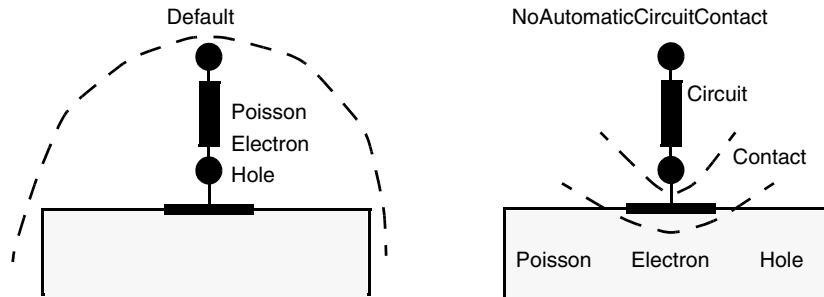
#### Note:

Sentaurus Device does not add circuit and contact equations if `Poisson` is restricted to instances. For example:

```
Coupled { device1.Poisson device1.Electron device1.Hole
           device2.Poisson device2.Electron device2.Hole }
```

If the keyword `NoAutomaticCircuitContact` appears in the `Math` section, Sentaurus Device does not add the circuit and contact equations automatically (see [Figure 15](#)).

*Figure 15 Range of equation keywords Circuit, Contact, Poisson, Electron, and Hole*



## Selecting Individual Devices in Mixed Mode

The default usage of an equation keyword such as `Poisson` activates the given equations for all devices. With complex multiple-device systems, such an action is not always desirable especially when a fully consistent solution has not yet been found. Sentaurus Device allows each equation to be restricted to a specific device by adding the name of the device instance to the equation keyword separated with a period. The syntax is:

```
<identifier>.<equation>
```

Device-specific solutions are used to obtain the initial solution for the whole system. For example, with two or more devices, it is often better to solve each device individually before coupling them all. Such a scheme can be written as:

```
System {
    ... device1 ...
    ... device2 ...
}

Solve {
    # Solve Circuit equation Circuit

    # Solve poisson and full coupled for each device
    Coupled { device1.Poisson device1.Contact }
    Coupled { device1.Poisson device1.Contact
              device1.Electron device1.Hole }
    Coupled { device2.Poisson device2.Contact }
    Coupled { device2.Poisson device2.Contact
              device2.Electron device2.Hole }

    # Solve full coupled over all devices
    Coupled { Circuit Poisson Contact Electron Hole }
}
```

## Relaxed Newton Method

The relaxed Newton method allows Sentaurus Device to consider whether a solution that has neither converged nor diverged by the final Newton iteration should be accepted as a converged solution based on relaxed convergence criteria. This method applies to Quasistationary, Transient, and Continuation ramps.

Relaxed convergence criteria are established from relaxed Newton parameters that are specified by users in the `AcceptNewtonParameter` (ANP) statement of the global `Math` section. These parameters are used to calculate an *ANP RhsMin* and an *ANP error* that will be used to test for relaxed convergence:

```
Math {
    ...
    AcceptNewtonParameter (
        [RhsAndUpdateConvergence]      * Use -RhsAndUpdateConvergence
        [RhsMin=<float>]             * to relax convergence
                                    * The ANP RhsMin that is used
```

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

```
[UpdateScale=<float>] * for relaxed convergence
[RelErrControl] * This multiplies update error
[Digits=<integer>] * to give a relaxed ANP error
[ErrRef ( * Specifies the error control method
          * for calculating the ANP error
          * The number of relative error digits
          * in the ANP error calculation
          * Reference error parameters to use
          * in the ANP error calculation
          [Poisson=<float>] [Electron=<float>] [Hole=<float>]
          [Temperature=<float>] [eTemperature=<float>]
          [hTemperature=<float>]
          [eQuantumPotential=<float>] [hQuantumPotential=<float>]
        )]
[Error ( * Absolute error parameters to use
          * in the ANP error calculation
          [Poisson=<float>] [Electron=<float>] [Hole=<float>]
          [Temperature=<float>] [eTemperature=<float>]
          [hTemperature=<float>]
          [eQuantumPotential=<float>] [hQuantumPotential=<float>]
        )]
[InvokeAtDivergence] * Allows relaxed convergence for
                      * diverging solutions
      )
}
```

To activate the relaxed Newton method, specify `AcceptNewtonParameter` in the `Quasistationary`, `Transient`, or `Continuation` statements of the `Solve` section, along with a `ReferenceStep` (except for `Continuation`).

For example:

```
Solve { ...
  Quasistationary ( ...
    AcceptNewtonParameter ( ReferenceStep = 1.e-3 )
  ) { Coupled { ... } }
}
```

or:

```
Solve { ...
  Transient ( ...
    AcceptNewtonParameter ( ReferenceStep = 1.e-9 )
  ) { Coupled { ... } }
}
```

or:

```
Solve { ...
  Continuation ( ...
    AcceptNewtonParameter
  ) { Coupled { ... } }
}
```

## Chapter 6: Numeric and Software-Related Issues

### Nonlinear Iterations

During the simulation, if the Quasistationary or Transient step size is reduced below the value specified with ReferenceStep, Sentaurus Device will begin to calculate an ANP error in addition to the standard update error for each Newton iteration during a solution. This will be used to determine whether relaxed convergence has been achieved if the solution fails to converge using the standard error criteria. For Continuation, the check for relaxed convergence is activated for all step sizes.

Relaxed convergence is considered achieved if the calculated ANP error < 1 or the RHS norm < ANP RhsMin, for any iteration. If relaxed convergence is achieved, Sentaurus Device will store the solution obtained from the iteration with the smallest standard update error occurring after relaxed convergence is triggered. In most cases, this is a reasonable choice for the “best” solution.

Note the following:

- The relaxed Newton parameters are passed only to the next-level Coupled statements, that is, Coupled statements in Plugin statements do not use the relaxed Newton parameters.
- Only specified parameter values are passed to the Coupled statement, that is, no default values are used.

---

## Plugin Command

The Plugin command controls an iterative loop over two or more Coupled statements. It is used when a fully coupled method would use too many resources of a given machine, or when the problem is not yet solved and a full coupling of the equations would diverge.

The Plugin syntax is defined as:

```
Plugin ( <options> ) ( <list-of-coupled-commands> )
```

Plugin commands can have any complexity but, usually, only a few combinations are effective. One standard form is the Gummel iteration in which each basic semiconductor equation is solved consecutively.

With the Plugin command, this is written as:

```
Plugin { Poisson Electron Hole }
```

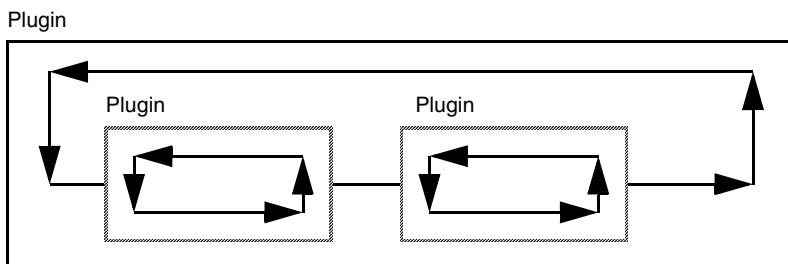
[Table 347 on page 1729](#) summarizes the options of Plugin.

Plugin commands can be used with other Plugin commands. For example:

```
Plugin{ Plugin{ ... } Plugin { ... } }
```

[Figure 16](#) illustrates the corresponding loop structure. A hierarchy of Plugin commands allows more complex iterative solve patterns to be created.

*Figure 16 Example of hierarchy of Plugin commands*



## Linear Solvers

The `Math` parameters to the solution algorithms are device independent and must appear only in the base `Math` section. These can be grouped by solver type. The control parameters for the linear solvers are `Method` and `SubMethod`. The keyword `Method` selects the linear solver to be used, and the keyword `SubMethod` selects the inner method for block-decomposition methods (see [Table 223 on page 1610](#) for available linear solvers). The default value for `Method` is the blocked decomposition solver `Blocked`. The default value for `SubMethod` is `Super` for 2D devices and `ILS(set=1)` for 3D devices.

The keywords `ACMethod` and `ACSubMethod` determine the linear solver used for AC analysis.

### Note:-

`ACMethod=Blocked` is the only valid choice for `ACMethod`. However, any of the available linear solvers can be selected for `ACSSubMethod`.

[Table 223](#) lists available options for the linear solver PARDISO. The options are specified in parentheses after the solver specification:

```
Method = ParDiSo (NonsymmetricPermutation IterativeRefinement)
```

The default options `NonsymmetricPermutation`, `-IterativeRefinement`, and `-RecomputeNonsymmetricPermutation` provide the best compromise between speed and accuracy. To improve speed, select `-NonsymmetricPermutation`.

To improve accuracy, at the expense of speed, activate `IterativeRefinement`, or `RecomputeNonsymmetricPermutation`, or both.

All ILS options can be specified within an `ILSrc` statement in the global `Math` section:

```
Math {
    ILSrc = "
        set (...) {
            iterative (...);
            preconditioning (...);
            ordering (...);
```

## Chapter 6: Numeric and Software-Related Issues

### Nonlocal Meshes

```
        options ( . . . );
    } ;
    . . .
    "
    . . .
}
```

See [Table 223 on page 1610](#) for additional ILS options.

The linear solvers PARDISO and ILS support the option `MultipleRHS` to solve linear systems with multiple right-hand sides. This option is only appropriate for AC analysis. ILS might produce a small parallel speedup or slightly more accurate results if this option is selected.

Use the `Math` option `PrintLinearSolver` to obtain additional information regarding the linear solver being used.

For 1D and 2D structures, the linear solver SUPER is used by default. For 3D structures, ILS is used by default with set (1).

---

## Nonlocal Meshes

Nonlocal meshes are one-dimensional, special-purpose meshes that Sentaurus Device needs to implement one-dimensional, nonlocal physical models. A nonlocal mesh consists of nonlocal lines. Each nonlocal line is subdivided by nonlocal mesh points, to allow for the discretization of the equations that constitute the physical models. Sentaurus Device performs this subdivision automatically to obtain optimal interpolation between the nonlocal mesh and the normal mesh.

The 1D Schrödinger equation (see [1D Schrödinger Equation on page 351](#)), the nonlocal tunneling model (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 826](#)), and the trap tunneling model (see [Nonlocal Tunneling for Traps on page 576](#)) use nonlocal meshes. The documentation of the first two models introduces the use of nonlocal meshes in the context of the particular model and is restricted to typical cases. This section describes the construction of nonlocal meshes in detail.

---

## Specifying Nonlocal Meshes

You specify nonlocal meshes using the keyword `Nonlocal`, in the global `Math` section, followed by a string that gives the name of the nonlocal mesh and a list of options that control the construction of the nonlocal mesh. You can specify any number of nonlocal meshes; individual specifications are independent.

## Chapter 6: Numeric and Software-Related Issues

### Nonlocal Meshes

In the following example, Sentaurus Device constructs a nonlocal mesh named `GateNonLocalMesh` that consists of nonlocal lines for semiconductor vertices up to a distance of 5 nm from the `Gate` electrode:

```
Math {
    Nonlocal "GateNonLocalMesh" (
        Electrode="Gate"
        Length=5e-7
    )
}
```

In the following example, Sentaurus Device constructs a nonlocal mesh named `for_tunneling` that consists of lines that connect the different sides of the region `gateoxide`:

```
Math {
    Nonlocal "for_tunneling" (
        Barrier(Region="gateoxide")
    )
}
```

For a summary of available options, see [Table 226 on page 1612](#). For a comprehensive description of the construction of a nonlocal mesh, see [Constructing Nonlocal Meshes on page 204](#).

---

## Visualizing Nonlocal Meshes

Sentaurus Device can visualize the nonlocal meshes it constructs. This feature is used to verify that the nonlocal mesh constructed is the one actually intended. For visualizing data defined on nonlocal meshes, see [Visualizing Data Defined on Nonlocal Meshes on page 204](#).

To visualize nonlocal meshes, use the keyword `NonLocal` in the `Plot` section (see [Device Plots on page 177](#)). The keyword causes Sentaurus Device to write two vector fields to the plot file that represent the nonlocal meshes constructed in the device.

For each vertex (of the normal mesh) for which a nonlocal line exists, the first vector field `NonLocalDirection` contains a vector that points from the vertex to the end of the nonlocal line in the direction of the reference surface for which the nonlocal line was constructed. The vector in the second field `NonLocalBackDirection` points from the vertex to the other end of the nonlocal line. The unit of both vectors is micrometer.

For vertices for which no nonlocal line exists, both vectors are zero. For vertices for which more than one nonlocal line exists, Sentaurus Device plots the vectors for one of these lines.

## Visualizing Data Defined on Nonlocal Meshes

To visualize data defined on nonlocal meshes:

- In the `File` section, specify a file name using the `NonLocalPlot` keyword.
- On the top level of the command file, specify a `NonLocalPlot` section. There, `NonLocalPlot` is followed by a list of coordinates in parentheses and a list of datasets in braces.

Sentaurus Device writes nonlocal plots at the same time it writes normal plots. Nonlocal plot files have the extension `.plt` or `.tdr`.

Sentaurus Device picks nonlocal lines close to the coordinates specified in the `NonLocalPlot` section for output. The datasets given in the `NonLocalPlot` section are the datasets that can be used in the `Plot` section (see [Device Plots on page 177](#)).

`NonLocalPlot` does not support the option `/Vector`. In addition, the Schrödinger equation provides special-purpose datasets available only for `NonLocalPlot` (see [Visualizing the Solutions on page 355](#)).

In addition to the datasets explicitly specified, Sentaurus Device automatically includes the `Distance` dataset in the output. It provides the coordinate along the nonlocal line. The values in the `Distance` dataset are measured in  $\mu\text{m}$ . The interface or contact for which a nonlocal mesh line was constructed is located at zero, and its mesh vertex is located at positive coordinates.

The following example plots electron and hole densities for the nonlocal lines close to the coordinates  $(0, 0, 0)$  and  $(0, 1, 0)$  in the device:

```
NonLocalPlot(
    (0 0) (0 1)
){  
    eDensity hDensity
}
```

---

## Constructing Nonlocal Meshes

Nonlocal meshes are specified in the global `Math` section:

```
Math {
    NonLocal <string> (
        Barrier(...)
        RegionInterface=<string>
        MaterialInterface=<string>
        Electrode=<string>
        ...
    )
}
```

## Chapter 6: Numeric and Software-Related Issues

### Nonlocal Meshes

The string following `NonLocal` is the name of the nonlocal mesh. The name relates the nonlocal mesh to the physical models defined on it.

Sentaurus Device supports two ways to specify nonlocal meshes:

- Specify the regions and materials that form the tunneling barrier (keyword `Barrier`).
- Specify a reference surface (keywords `RegionInterface`, `MaterialInterface`, and `Electrode`).

The `Barrier` specification is simpler but less general, and it is only suitable for nonlocal meshes used for direct tunneling through barriers that are assumed to be insulating.

### Specification Using Barrier

As an option to `Barrier`, specify all regions that belong to the tunneling barrier, using any number of `Region=<string>` or `Material=<string>` specifications. Sentaurus Device connects each side of the barrier to any other with nonlocal lines. Here, `side` means a conductively connected part of the surface of the barrier.

By default, semiconductor regions, metal regions, and electrodes are considered to be conductive. To enforce that a particular region (such as a wide-bandgap semiconductor region) is treated as not conductive, use the option `-Endpoint`, which accepts as an option a list of regions and materials, using the same syntax as for `Barrier`.

The following example causes regions `buffer1` and `buffer2`, and all GaN regions to be considered nonconductive when constructing the nonlocal mesh `NLM`:

```
Nonlocal "NLM" (
    ...
    -Endpoint(Material="GaN" Region="buffer1" Region="buffer2")
)
```

Use `Length=<float>` (in centimeters) to restrict the length on nonlocal lines (to suppress very long tunneling paths).

### Specification Using a Reference Surface

`RegionInterface`, `MaterialInterface`, and `Electrode` specify a region interface name, material interface name, or electrode name. All interfaces and electrodes together form the reference surface that determines where in the device the nonlocal mesh is constructed.

The nonlocal lines link vertices of the normal mesh to the reference surface on the geometrically shortest path. The parameter `Length` of `NonLocal` determines the maximum distance of the vertex to the reference surface (in centimeters). Sentaurus Device provides no default value for `Length`; all nonlocal meshes must specify `Length` explicitly.

`Length` and `Permeation` are limited to the value of the parameter `NonLocalLengthLimit`, which is specified in centimeters in the global `Math` section and defaults to  $10^{-4}$ . This

## Chapter 6: Numeric and Software-Related Issues

### Nonlocal Meshes

parameter is used to capture cases where users accidentally specify lengths in micrometers rather than centimeters.

The property that nonlocal lines connect a vertex to the reference surface on the geometrically shortest path is fundamental. If any of the other rules described in this section inhibits the construction of a nonlocal line for this path, but a longer connection obeys all these restrictions, Sentaurus Device still does not use this connection to construct an alternative nonlocal line.

The parameter `Permeation` specifies a length by which Sentaurus Device extends the nonlocal lines, across the reference surface, towards the opposite of the side for which the line is constructed. `Permeation` defaults to zero. Sentaurus Device never extends the lines outside the device or into regions flagged with `-Permeable` (see below). The extension is not affected by the `Transparent` and `Endpoint` options (see below).

The `Direction` parameter specifies a direction that the nonlocal lines approximately should have. Nonlocal lines with directions that deviate from the specified direction by an angle greater than `MaxAngle` are suppressed. If `Direction` is the zero vector or `MaxAngle` exceeds 90 (this is the default), nonlocal lines can have any direction. The length and sign of the direction vector are otherwise insignificant.

`Discretization` specifies the maximum spacing of the nonlocal mesh vertices. When necessary, Sentaurus Device further refines the mesh created on a nonlocal line according to the built-in rules to yield a spacing no bigger than `Discretization` demands.

The flags `Transparent`, `Permeable`, `Endpoint`, and `Refined`, as well as their negation, specify flags for regions or materials that control nonlocal mesh construction. Each of the flags can be specified in two different ways:

- As an option to the main specification in the global `Math` section
- As an option to `NonLocal` in a region-specific or material-specific `Math` section

In the former case, the flag is followed by a list of region or material specifications that determine where the flag applies. In the latter case, the region or material of applicability is the location of the `Math` specification. Specifications of the latter style provide defaults that can be overwritten by specifications of the former style.

The part of a nonlocal line between the mesh vertex for which the line is constructed and the reference surface must not pass through regions flagged with `-Transparent`. By default, all regions are `Transparent`. For the exterior of the device, the flag `Outside` has the same meaning; this flag is an option to the main `NonLocal` specification.

The `-Permeable` flag limits extension on nonlocal lines across the reference surface. By default, all regions are `Permeable`.

For regions flagged with `-Endpoint`, Sentaurus Device does not construct nonlocal lines that end in this region. The default is `-Endpoint` for insulator regions, and `Endpoint` for other regions.

Inside regions flagged as `-Refined`, the generation of nonlocal mesh points at element boundaries is suppressed. By default, all regions are `Refined`.

---

## Special Handling of the 1D Schrödinger Equation

For performance reasons, Sentaurus Device solves the 1D Schrödinger equation (see [1D Schrödinger Equation on page 351](#)) only on a reduced subset of nonlocal lines that still cover all vertices of the normal mesh for which nonlocal lines are constructed according to the rules outlined above.

To avoid artificial geometric quantization, Sentaurus Device extends nonlocal lines used for the 1D Schrödinger equation that are shorter than `Length` to reach full length. Sentaurus Device never extends the lines outside the device or into regions flagged by the `-Permeable` option. The extension is not affected by the `Transparent` and `Endpoint` options.

For nonlocal line segments in regions not marked by `-Refined` and `-Endpoint`, Sentaurus Device computes the intersections with the boxes of the normal mesh. Sentaurus Device needs this information to interpolate results from the 1D Schrödinger equation back to the normal mesh. Therefore, in regions where `-Refined` or `-Endpoint` is specified, 1D Schrödinger density corrections are not available, even when the regions are covered by nonlocal lines.

---

## Special Handling of the Nonlocal Tunneling Model

Sentaurus Device computes the intersections of the nonlocal lines with the boxes (see [Discretization on page 1177](#)). To this end, some lines might become longer than `Length`. The nonlocal tunneling model uses the intersection points to limit the spatial range of the integrations Sentaurus Device must perform to compute the contribution to tunneling that comes from the particular vertex (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 826](#)). For mesh points that border regions flagged with `-Endpoint`, the integration range is extended into that region, to pick up the Fowler–Nordheim current that enters it.

For nonlocal meshes used only for nonlocal tunneling, when `Permeation` is zero, Sentaurus Device assumes that the nonlocal lines cross the reference plane by an infinitesimal length. Therefore, if `-Endpoint` is specified for one of the regions that border the reference plane, Sentaurus Device suppresses nonlocal lines that point into that region.

If `Permeation` is positive, for nonlocal lines at nonmetal interfaces, Sentaurus Device switches to a nonlocal mesh construction mode similar to the one used for the Schrödinger equation: Sentaurus Device constructs only as many lines as needed to cover all vertices that must be covered, and extends all those lines to their maximum length. Furthermore, for nonlocal meshes not used for tunneling to traps, the integration range for tunneling covers the entire line.

## Chapter 6: Numeric and Software-Related Issues

### Nonlocal Meshes

Tunneling to and from segments of the line marked by `-Endpoint` or `-Refined` will be assigned to the vertices for boxes crossed immediately outside those segments, to pick up Fowler–Nordheim currents in a similar manner as for the line construction mode with `Permeation=0`.

---

## Unnamed Meshes

For backward compatibility, you can specify nonlocal meshes in interface-specific or electrode-specific `Math` sections. For this kind of specification, the location of the mesh is the location of the `Math` section and, therefore, no interface or electrode specification must appear as an option to `NonLocal`.

Furthermore, the name of the nonlocal mesh must be omitted. Physical models are associated with the nonlocal mesh by activating them in a `Physics` section specific to the same interface or electrode for which the nonlocal mesh was constructed.

For unnamed nonlocal meshes, vertices in regions with the trap tunneling model (see [Nonlocal Tunneling for Traps on page 576](#)) are connected irrespective of the material of the region or the `Endpoint` specification.

---

## Performance Suggestions

To limit the negative performance impact of the nonlocal tunneling model, it is important to limit the number of nonlocal lines. To this end, most importantly, select `Length` and `Permeation` to be only as long as necessary. The option `-Endpoint` can be used to suppress the construction of lines to regions for which you know, in advance, that will not receive much tunneling current. The option `-Transparent` allows you to neglect tunneling through materials with comparatively high tunneling barrier, for example, oxides near a Schottky contact or heterointerface for which nonlocal tunneling has been activated.

Another use of the option `-Transparent` is at heterointerfaces, where there is no tunneling to the side of the material with the lower band edge (as there is no barrier to tunnel through). To restrict the construction of nonlocal lines to lines that go through the larger band-edge material, declare the lower band-edge material as not transparent by using `-Transparent`.

The option `-Refined` does not reduce the number of nonlocal lines, but it can reduce the size of the Jacobian matrix. The option `-Refined` is most useful for insulator regions, where the band-edge profile is approximately linear.

## Monitoring Convergence Behavior

When Sentaurus Device has convergence problems, it can be helpful to know in which parts of the device and for which equations the errors are particularly large. With this information, it is easier to adjust the mesh or the models used, to improve convergence.

Sentaurus Device can print the locations in the device where the largest errors occur (see [CNormPrint on page 209](#)). This provides limited information and has negligible performance impact. Sentaurus Device can also plot solution error information for the entire device after each Newton step (see [NewtonPlot on page 209](#)). This information is comprehensive, but can generate many files and can take significant time to write.

Both approaches provide access to the internal data of Sentaurus Device. Therefore, in both cases, the output is implementation dependent. Its proper interpretation can change between different Sentaurus Device releases.

---

### CNormPrint

To obtain basic error information, specify the `CNormPrint` keyword in the global `Math` section. Then, after each Newton step and for each equation solved, Sentaurus Device prints the following to standard output:

- Largest error according to [Equation 33 on page 194](#) that occurs anywhere in the device for the equation
  - Vertex where the largest error occurs
  - Coordinates of the vertex
  - Current value of the solution variable for that vertex
- 

### NewtonPlot

Sentaurus Device can write the spatial values of solution variables, the errors, the right-hand sides, and the solution updates to a `NewtonPlot` file after each Newton step. `NewtonPlot` files then can be read into Sentaurus Visual for examination. To use this feature:

- Use the `NewtonPlot` keyword in the `File` section to specify a file name for the plot. This name can contain up to two C-style integer format specifiers (for example, `%d`). If present, for the file name generation, the first one is replaced by the iteration number in the current Newton step and the second, by the number of Newton steps in the simulation so far. Sentaurus Device does not enforce any particular file name extension, but prepends the device instance name to the file name in mixed mode.

## Chapter 6: Numeric and Software-Related Issues

### Monitoring Convergence Behavior

- Sentaurus Device writes Newton information to a `NewtonPlot` file during execution of a `Quasistationary` or `Transient` command when the step size decreases below a certain value. By default, the criterion is `step size < 2 × MinStep`, where `MinStep` is the lower limit for the step size that is set in a `Quasistationary` or `Transient` command. This condition usually occurs immediately before a simulation is about to fail.
- Alternatively, use the `NewtonPlotStep` parameter to specify the step-size criterion. `NewtonPlotStep` can be specified as an option to the `NewtonPlot` keyword in the `Math` section of the command file, in which case, it applies to all `Quasistationary` and `Transient` commands. It also can be specified as an option to a particular `Quasistationary` or `Transient` command, in which case, it only applies to that particular command.
- By default, when the step-size criterion is met, Sentaurus Device writes the current values of the solution variables only to the `NewtonPlot` file. Additional data can be included in the output by specifying options to `NewtonPlot` in the `Math` section. See [Table 211 on page 1573](#) for a list of available options.
- In addition, `MinError` can be specified as an option to `NewtonPlot`. If present, Sentaurus Device writes the `NewtonPlot` file only if the error of the actual iteration is smaller than all previous errors within the current Newton step. In this case, the first `%d` format specifier in the `NewtonPlot` file name will be replaced with `min` instead of the iteration number.

---

## Automatic Activation of `CNormPrint` and `NewtonPlot`

By default, `CNormPrint` and `NewtonPlot` are activated automatically when certain criteria are met:

- For `CNormPrint`: `step size < AutoCNPMinStepFactor × MinStep`
- For `NewtonPlot`: `step size < AutoNPMMinStepFactor × MinStep`

where the `step size` is calculated during the execution of a `Quasistationary` or `Transient` statement, and `AutoCNPMinStepFactor` and `AutoNPMMinStepFactor` are parameters that can be specified in the `Math` section of the command file:

```
Math {  
    AutoCNPMinStepFactor = <float>    #default = 2.0  
    AutoNPMMinStepFactor = <float>    #default = 2.0  
}
```

You can deactivate the automatic activation of `CNormPrint` and `NewtonPlot` by specifying values of zero for the above parameters.

When `NewtonPlot` files are created as a result of automatic activation, the `Error`, `Residual`, `Update`, and `MinError` options for `NewtonPlot` will be used by default. However,

## Chapter 6: Numeric and Software-Related Issues

### Monitoring Convergence Behavior

any user-specified options for `NewtonPlot` in the `Math` section of the command file will override the default behavior.

In addition, `NewtonPlot` will be activated automatically for solutions that are not part of a `Quasistationary` or `Transient` when the maximum-allowed iterations for the solution have been reached. The same output options described in the previous paragraph will be used in this case as well (except for `MinError`), and the iteration number will replace the `%d` format specifier in the file name.

---

## Simulation Statistics for Plotting and Output

This section discusses simulation statistics.

### Simulation Statistics in Current Plot Files

To include various simulation statistics in current plot files for visualization, specify the `SimStats` option in the `Math` section of the command file:

```
Math {SimStats}
```

When this option is specified, Sentaurus Device writes a `SimStats` group of datasets to the current plot file (`*.plt` file) after each successful solution. The following datasets are included in the `SimStats` group:

<code>AssemblyTime</code>	CPU time spent building RHS and Jacobian for the solution
<code>AssemblyTime_wall</code>	Wallclock time spent building RHS and Jacobian for the solution
<code>AverageMemory</code>	Average memory for the time step
<code>CumulativeAssemblyTime</code>	Total assembly time so far
<code>CumulativeAssemblyTime_wall</code>	Total wallclock assembly time so far
<code>CumulativeIterations</code>	Total number of iterations so far
<code>CumulativeRestarts</code>	Total number of restarts so far
<code>CumulativeSolveTime</code>	Total solve time so far
<code>CumulativeSolveTime_wall</code>	Total wallclock solve time so far
<code>CumulativeTotalTime</code>	Total CPU time so far
<code>CumulativeTotalTime_wall</code>	Total wallclock time so far

## Chapter 6: Numeric and Software-Related Issues

### Monitoring Convergence Behavior

error	Final error for the solution
Iterations	Number of iterations required for solution
PeakMemory	Peak memory (watermark) so far
Restarts	Number of consecutive restarts before the solution converged
Rhs	Final RHS norm for the solution
SolveTime	CPU time used by solver for the solution
SolveTime_wall	Wallclock time used by solver for the solution
Stepsize	Step size used for quasistationary or transient solution
TotalTime	Total CPU time for the solution (includes overhead time)
TotalTime_wall	Total wallclock time for the solution (includes overhead time)

## Simulation Statistics in Design-of-Experiments Variables

Cumulative simulation statistics can be written to the end of a Sentaurus Device output file (\*.log file) as design-of-experiments (DOE) variables by specifying the `WriteDOE` option to `SimStats` in the `Math` section:

```
Math {
    SimStats ( WriteDOE [DOE_prefix = <string>] )
}
```

When `WriteDOE` is specified, the following DOE variables will be written:

```
DOE: Restarts      <value>
DOE: Iterations    <value>
DOE: AssemblyTime  <value>
DOE: SolveTime     <value>
DOE: TotalTime     <value>
DOE: AssemblyTime_wall <value>
DOE: SolveTime_wall <value>
DOE: TotalTime_wall <value>
```

The option `DOE_prefix` allows a string to be prepended to the DOE variable names. For example, specifying `DOE_prefix= "test_"` writes the DOE variable `test_Restarts` instead of `Restarts`.

## Save and Load Statements

The `Save` and `Load` statements allow you to store the current simulation state of a device in a file and later (often, from another simulation run) to reload it, and to resume from where you stopped. By default, `Plot` files can be reloaded as well (see [Device Plots on page 177](#)).

The `Save` statement generates files of type `.sav`, which only contain the information required to restart a simulation such as:

- Contact biases and currents
- Values of solution variables
- Occupation probability of traps
- Ferroelectric history (electric field and polarization) if the ferroelectric model is switched on

Loading traps is supported only if the specifications of traps for the saving and loading simulation are identical, that is, the set of traps and their order in the command file must be identical, as well as their types. A mismatch of trap specifications is silently ignored and could lead to partially correctly, incorrectly, or not loaded trap occupations. In this case, you must verify if the traps are initialized as intended.

When the file is loaded, all this information is restored.

**Note:**

Contact voltages stored in the loaded file overwrite the bias conditions that are specified in the command file.

When you specify a `Save` file in the `File` section, the simulation state is saved automatically to that file at the end of the simulation. When you specify a `Load` file in the `File` section, the state stored in that file is loaded at the beginning of the simulation. The geometry of a loaded file must match the geometry specified in the `Grid` file.

**Note:**

Interfaces regions are required in the grid file to save and load data on interfaces (see [Interface Plots on page 180](#)).

More control is possible with `Save` and `Load` commands in the `Solve` section. The commands are defined as:

```
Save(<parameters-opt>) <system-opt>
Load(<parameters-opt>) <system-opt>
```

The options are as for `Plot` (see [When to Plot on page 178](#)). `Save` statements can be used at any level of the `Solve` section, and the `Load` statement can be used only as a base level

## Chapter 6: Numeric and Software-Related Issues

### Tcl Command File

of the `Solve` section. For example, in the case of a transient simulation, `Load` can be used only before or after the transient, not during it.

Multiple `Load` and `Save` commands are allowed in a `Solve` section. For example:

```
Solve {  
    ...  
    # Ramp the gate and save structures  
    # First gate voltage  
    Quasistationary (InitialStep=0.1 MaxStep=0.1 MinStep=0.01  
        Goal {Name="gate" Voltage=1})  
        {Coupled {Poisson Electron Hole}}  
        Save(FilePrefix="vg1")  
    # Second gate voltage  
    Quasistationary (InitialStep=0.1 MaxStep=0.1 MinStep=0.01  
        Goal {Name="gate" Voltage=3})  
        {Coupled {Poisson Electron Hole}}  
        Save(FilePrefix="vg2")  
    # Load the saved structures and ramp the drain  
    # First curve  
    Load(FilePrefix="vg1")  
    NewCurrentPrefix="Curve1"  
    Quasistationary (InitialStep=0.1 MaxStep=0.5 MinStep=0.01  
        Goal {Name="drain" Voltage=2.0})  
        {Coupled {Poisson Electron Hole}}  
    # Second curve  
    Load(FilePrefix="vg2")  
    NewCurrentPrefix="Curve2"  
    Quasistationary (InitialStep=0.1 MaxStep=0.5 MinStep=0.01  
        Goal {Name="drain" Voltage=2.0})  
        {Coupled {Poisson Electron Hole}}  
}
```

---

## Tcl Command File

The Sentaurus Device command-line option `--tcl` invokes the Tcl interpreter to execute the command file. Tcl provides valuable extensions to the standard command file of Sentaurus Device, including:

- Variables
- Expressions
- Control structures

In addition, all of the Inspect Tcl batch commands are available (see the *Inspect User Guide*). These commands are useful for parameter extraction (see [Extracting Parameters on page 218](#)).

## Performing Device Simulations

An entire device simulation can be performed using the Tcl command `sdevice`. The following example shows a quasistationary ramp to computed contact values:

```

set vd 0.2
set vg [expr {10*$vd}]

sdevice "
    Electrode{
        { Name = \"source\" Voltage = 0.0 }
        { Name = \"gate\" Voltage = 0.0 }
        { Name = \"drain\" Voltage = $vd }
        { Name = \"bulk\" Voltage = 0.0 }
    }

    File{
        Grid      = \"mosfet.tdr\"
        Plot     = \"mosfet_des.tdr\"
        Current  = \"mosfet_des.plt\"
        Output   = \"mosfet_des.log\"
    }

    Physics {
        Mobility (DopingDependence)
        Recombination (SRH (DopingDependence))
    }

    Solve{
        Poisson
        QuasiStationary (Goal {Name=\"gate\" Voltage=$vg})
            { Coupled {Poisson Electron Hole} }
    }
"

```

Sometimes, it is more useful to perform a device simulation in stages. In this case, the `sdevice_init` command is used for initialization, and multiple `sdevice_solve` commands are used to drive the simulation.

In the following example, an  $I_d$ - $V_g$  sweep is performed, and the threshold voltage is extracted from the current plot file:

```

# initialize Sentaurus Device simulation
sdevice_init {
    Electrode{
        { Name = "source" Voltage = 0.0 }
        { Name = "gate" Voltage = 0.0 }
        { Name = "drain" Voltage = 0.2 }
        { Name = "bulk" Voltage = 0.0 }
    }

    File{

```

## Chapter 6: Numeric and Software-Related Issues

### Tcl Command File

```
Grid      = "mosfet.tdr"
Plot      = "extract_VT_des.tdr"
Current   = "extract_VT_des.plt"
Output    = "extract_VT_des.log"
}

Physics{
    EffectiveIntrinsicDensity (Slotboom)
    Mobility (DopingDependence)
    Recombination (SRH (DopingDependence))
}
}

# compute idvgs curve
sdevice_solve {
    Solve{
        NewCurrentPrefix = "initial_"

        Poisson
        Coupled {Poisson Electron Hole}

        Save (FilePrefix = "extract_VT_initial_save")

        NewCurrentPrefix = ""
        QuasiStationary (
            InitialStep=0.05 Increment=1.3
            MaxStep=0.05 MinStep=1e-4
            Goal {Name="gate" Voltage=2}
        )
        { Coupled {Poisson Electron Hole} }
    }
}

# extract threshold voltage
set VT [extract_VT extract_VT_des.plt gate drain]

# ramp gate voltage to threshold voltage
sdevice_solve "
    Solve{
        NewCurrentPrefix = \"ignore_\"

        Load (FilePrefix = \"extract_VT_initial_save\")

        QuasiStationary (
            InitialStep=0.25 Increment=1.3
            MaxStep=0.25 MinStep=1e-4
            Goal {Name=\"gate\" Voltage=$VT}
        )
        { Coupled {Poisson Electron Hole} }
    }
"
"
```

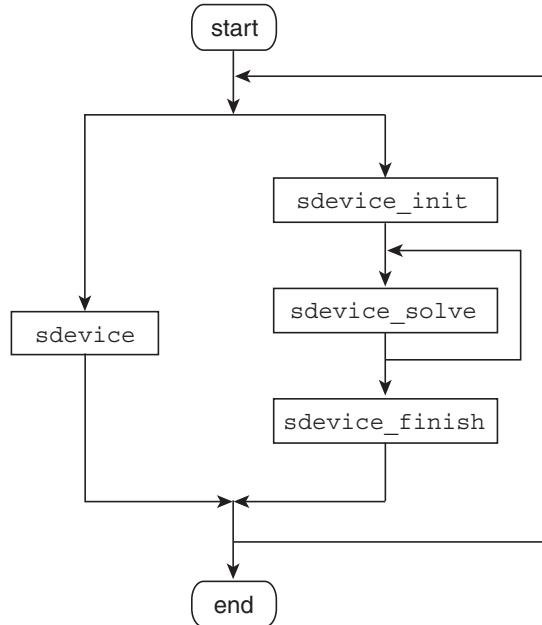
## Chapter 6: Numeric and Software-Related Issues

### Tcl Command File

```
# save final plot file  
sdevice_finish
```

The final command `sdevice_finish` signals the end of a sequence of `sdevice_solve` statements, and Sentaurus Device saves the final plot files.

*Figure 17 Flowchart showing sequence in which various Tcl commands are invoked*



## **sdevice Command**

This Tcl command expects one argument, which describes an entire device simulation, that is, it includes a `File` section, `Physics` section, and `Solve` section.

The `sdevice` command returns 1 if the device simulation converges. Otherwise, the value 0 is returned.

## **sdevice\_init Command**

This Tcl command expects one argument, which initializes a device simulation, that is, it can contain all of the sections of a standard command file of Sentaurus Device (except a `Solve` section).

No return value is computed by `sdevice_init`. If an error is encountered, Sentaurus Device will terminate.

## **sdevice\_solve Command**

This Tcl command expects one argument and can be used only after an `sdevice_init` command. The argument must contain a single `Solve` section. All the commands in the `Solve` section are executed.

The `sdevice_solve` command returns 1 if the device simulation converges. Otherwise, the value 0 is returned.

## **sdevice\_finish Command**

This Tcl command expects no arguments and can be called after a series of `sdevice_solve` commands. It indicates that the device simulation performed by the `sdevice_solve` commands is finished, and the final plot file is generated. All open current plot files are closed.

## **sdevice\_parameters Command**

This Tcl command expects two arguments: a file name and the contents of a Sentaurus Device parameter file. It is an auxiliary function, which writes the parameter file to the given file name.

The following example generates the file `models.par` containing a permittivity parameter:

```
sdevice_parameters models.par {
    Epsilon {
        epsilon = 11.7
    }
}
```

---

## **Extracting Parameters**

The Tcl interpreter in Sentaurus Device provides access to all of the `Inspect` commands. In this way, you can easily analyze `.plt` files generated by Sentaurus Device and extract parameters. The following example shows a gate ramp followed by the extraction of the threshold voltage:

```
sdevice {
    Electrode {
        { Name = "source" Voltage = 0.0 }
        { Name = "gate"   Voltage = 0.0 }
        { Name = "drain"  Voltage = 0.2 }
        { Name = "bulk"   Voltage = 0.0 }
    }
    File {
        Grid = "mosfet.tdr"
        Current = "IdVg_des.plt"
    }
}
```

## Chapter 6: Numeric and Software-Related Issues

### Tcl Command File

```
Physics { ... }
Solve {
    QuasiStationary (
        InitialStep=0.05 MaxStep=0.05
        Goal {Name="gate" Voltage=2}
    ) { Coupled {Poisson Electron Hole} }
}
load_library EXTRACT
proj_load IdVg_des.plt iv
cv_create idvgs "iv gate OuterVoltage" "iv drain TotalCurrent"
set Vtgm [ExtractVtgm Vtgm idvgs]
set Vti [ExtractVti Vti idvgs 1e-8]
set gm [ExtractGm gm idvgs]
set SS [ExtractSS SS idvgs 1e-2]
```

## Available Inspect Tcl Commands

Inspect Tcl commands can be used in Sentaurus Device.

Table 21     *Inspect Tcl commands available for use in Sentaurus Device*

Command functions	Tcl commands
Reading and writing files	graph_load, graph_write, load_library, param_load, param_write, proj_getDataSet, proj_getList, proj_getNodeList, proj_load, proj_unload, proj_write
Curve commands	cv_abs, cv_compute, cv_create, cv_createFromScript, cv_createWithFormula, cv_delete, cv_delPts, cv_getVals, cv_getValsX, cv_getValsY, cv_getXaxis, cv_getYaxis, cv_getZero, cv_inv, cv_log10Scale, cv_logScale, cv_printVals, cv_rename, cv_renameCurve, cv_reset, cv_scaleCurve, cv_set_interp, cv_split, cv_split_disc, cv_write, macro_define
Parameter extraction	f_Gamma, f_gm, f_IDSS, f_KP, f_Ron, f_Rout, f_TetaG, f_VT, f_VT1, f_VT2, ft_scalar
Curve comparison library (load_library curvecomp)	cvcmp_CompareTwoCurves, cvcmp_DeltaTwoCurves
Extraction library (load_library EXTRACT)	cv_linTransCurve, cv_scaleCurve, ExtractBVi, ExtractBVv, ExtractEarlyV, ExtractGm, ExtractGmb, ExtractIoff, ExtractMax, ExtractRon, ExtractSS, ExtractValue, ExtractVtgm, ExtractVtgmb, ExtractVti

## Chapter 6: Numeric and Software-Related Issues

### Tcl Command File

Table 21     *Inspect Tcl commands available for use in Sentaurus Device (Continued)*

Command functions	Tcl commands
RF extraction library (load_library RFX)	rfx_abs_c, rfx_abs2_c, rfx_add_c, rfx_con_c, rfx_div_c, rfx_Export2csv, rfx_ExtractVal, rfx_GetFK1, rfx_GetFmax, rfx_GetFt, rfx_GetK_MSG_MAG, rfx_GetMUG, rfx_GetNearestIndex, rfx_GetRFCList, rfx_im_c, rfx_load, rfx_mag_phase, rfx_mul_c, rfx_PolarBackdrop, rfx_re_c, rfx_ReIm2Abs, rfx_ReIm2Phase, rfx_S2K, rfx_sign, rfx_SmithBackdrop, rfx_sub_c, rfx_Y2H, rfx_Y2K, rfx_Y2S, rfx_Y2U, rfx_Y2Z, rfx_Z2U
IC-CAP model parameter extraction library (load_library ise2iccap)	iccap_Write

## Redirecting Output

The Tcl commands `sdevice`, `sdevice_init`, `sdevice_solve`, and `sdevice_finish` allow you to redirect standard output and standard errors.

Table 22     *Syntax for output redirection*

Syntax	Description
<code>&gt; filename</code>	Writes standard output to <code>filename</code> .
<code>2&gt; filename</code>	Writes standard error to <code>filename</code> .
<code>&gt;&amp; filename</code>	Writes both standard output and standard error to <code>filename</code> .
<code>&gt;&gt; filename</code>	Appends standard output to <code>filename</code> .
<code>2&gt;&gt; filename</code>	Appends standard error to <code>filename</code> .
<code>&gt;&gt;&amp; filename</code>	Appends both standard output and standard error to <code>filename</code> .

## Known Restrictions

The following restrictions apply:

- If there is a `Set(TrapFilling=...)` statement inside an `sdevice_solve` command, then it applies only to subsequent statements within the surrounding `Solve` section (see

## Chapter 6: Numeric and Software-Related Issues

### Parallelization

[Explicit Trap Occupation on page 550](#)). This effect does not persist between multiple calls of the `sdevice_solve` command.

- If a transient simulation is performed using several `sdevice_solve` commands, then the correct initial time must be specified because Sentaurus Device does not remember the last transient time from the previous `sdevice_solve` command.

---

## Parallelization

Sentaurus Device uses thread parallelism to accelerate simulations on shared memory computers.

You can specify the number of threads in the global `Math` section of the Sentaurus Device command file:

```
Math { NumberOfThreads = number_of_threads }
```

*Table 23 Summary of Sentaurus Device components that have been parallelized*

Area	Description
Matrix assembly	<ul style="list-style-type: none"><li>• Computation of mobility, avalanche, current density, energy flux</li><li>• Assembly of Poisson, continuity, lattice, and temperature equations</li><li>• Modified local-density approximation (MLDA)</li><li>• Computation of box method coefficients</li></ul>
Linear solver	<ul style="list-style-type: none"><li>• Direct solver PARDISO</li><li>• Iterative solver ILS</li></ul>

You can specify a constant for the number of threads. For example:

```
NumberOfThreads = 2  
NumberOfThreads = maximum
```

Here, `maximum` is equivalent to the number of processors available on the execution platform.

If required, you can specify the number of threads separately for both the assembly and the linear solvers:

```
Math {  
    NumberOfAssemblyThreads = number_of_assembly_threads  
    NumberOfSolverThreads = number_of_solver_threads  
}
```

## Chapter 6: Numeric and Software-Related Issues

### Parallelization

**Table 24** summarizes the priority of the various methods for specifying the number of threads.

*Table 24 Specification of number of threads*

Priority	Matrix assembly	Linear solver
Highest	NumberOfAssemblyThreads	NumberOfSolverThreads
Lowest	NumberOfThreads	NumberOfThreads

The stack size per assembly thread can be specified in the global `Math` section of the Sentaurus Device command file:

```
Math { StackSize = stacksize_in_bytes }
```

Alternatively, the following UNIX environment variables are recognized:

```
SDEVICE_STACKSIZE, SNPS_STACKSIZE
```

#### Note:

By default, Sentaurus Device uses only one thread and the stack size is 1 MB.  
For most simulations, the default stack size is adequate.

In-line with other Sentaurus tools, you can also specify the number of threads and the maximum number of threads on the command line:

```
sdevice --threads <int> --max_threads <int> pp1234_des.cmd
```

The number of assembly threads and the number of solver threads are then set if the command-line option `--threads` is detected. The priority order for reading the number of threads is as follows:

1. Command-line specification using `--threads`
2. Threads specification in the `Math` section for the solver or the assembly process

In all cases, the number of threads is limited by the value of the `--max_threads` command-line option if it is detected.

Sentaurus Device requires one parallel license for every four threads. For example, a parallel simulation with 2–4 threads requires one parallel license, 5–8 threads require two licenses, 9–12 threads require three licenses, and so on.

In the `Math` section, you can specify the behavior if there are not enough parallel licenses available:

```
Math { ParallelLicense (option) }
```

## Chapter 6: Numeric and Software-Related Issues

### Extended Precision

Table 25 Available options if insufficient parallel licenses are available

Option	Description
Abort	Terminate the simulation.
Serial	(Default) Continue the simulation in serial mode.
Wait	Wait until enough parallel licenses become available.

#### Note:

These options apply only to the initial checkout of parallel licenses. Under certain circumstances, Sentaurus Device might later check in and check out parallel licenses as well. This might occur when another required license is temporarily unavailable. In this case, Sentaurus Device checks in all licenses that have been checked out so far (including parallel licenses), waits until all the required licenses become available, and checks them out again.

Observe the following recommendations to obtain the best results from a parallel Sentaurus Device run:

- Speedups are only obtained for sufficiently large problems. In general, the device grid should have at least 5000 vertices. Three-dimensional problems are good candidates for parallelization.
- It is sensible to run a parallel Sentaurus Device job on an empty computer. As soon as multiple jobs compete for processors, performance decreases significantly.
- Use the keyword `wallclock` in the `Math` section of the Sentaurus Device command file to display wallclock times rather than CPU times.
- Parallel execution produces different rounding errors. Therefore, the number of Newton iterations might change.

---

## Extended Precision

Sentaurus Device allows you to perform simulations using extended precision floating-point arithmetic. This option is switched on in the global `Math` section by the keyword `ExtendedPrecision`. [Table 26](#) lists the available options, the size of a floating-point number, and the precision.

## Chapter 6: Numeric and Software-Related Issues

### Extended Precision

Table 26 Extended precision floating-point arithmetic

Option	Description	Size	Precision
-ExtendedPrecision	double (default)	64 bits	$2.22 \times 10^{-16}$
ExtendedPrecision	long double	80 bits (Linux)	$1.08 \times 10^{-19}$
ExtendedPrecision(128)	double-double	128 bits	$4.93 \times 10^{-32}$
ExtendedPrecision(256)	quad-double	256 bits	$1.22 \times 10^{-63}$
ExtendedPrecision(Digits=<digits>)	arbitrary	$320 + 4.5<\text{digits}>$ bits	$10^{-<\text{digits}>}$

On Linux operating systems, 80-bit extended precision arithmetic is supported in hardware with no noticeable degradation in performance. However, the gain in accuracy compared to normal precision (19 decimal digits versus 16 decimal digits) is limited.

The datatypes double-double and quad-double are implemented in software on all platforms. They provide significant improvements in accuracy (32 and 63 decimal digits, respectively). However, the runtimes increase by a factor of 10 and 100, respectively.

Arbitrary precision supports floating-point arithmetic with an arbitrary number of digits. It can be used for small research problems where even quad-double might not provide sufficient accuracy, for example, ultrawide-bandgap semiconductors.

The storage requirements of arbitrary precision increase linearly with the number of digits. The runtimes, however, increase quadratically with the number of digits.

Extended precision can be a powerful tool to simulate wide-bandgap semiconductors, where it is necessary to resolve accurately small currents and very low carrier concentrations. It might or might not be beneficial for general convergence problems.

When using extended precision, the following points should be observed for the best results:

- In general, the direct linear solver SUPER provides the best accuracy. Additional choices include the direct linear solver PARDISO and the iterative linear solver ILS. Other linear solvers do not support extended precision.
- In the `Math` section:
  - Increase the value of `Digits`. Possible values are `Digits=15` for `ExtendedPrecision(128)` and `Digits=25` for `ExtendedPrecision(256)`. The value of `Digits` can be increased even further with arbitrary precision.

## Chapter 6: Numeric and Software-Related Issues

### System Command

- Decrease the value of `RhsMin`. Possible values are `RhsMin=1e-15` for `ExtendedPrecision(128)` and `RhsMin=1e-25` for `ExtendedPrecision(256)`. The value of `RhsMin` can be decreased even further with arbitrary precision.
- Slightly increase `Iterations`, for example, from 15 to 20.
- In the case of arbitrary precision, you might need to increase the maximum-allowed error (`UpdateMax` parameter) as well as to increase `Digits`. For example, `UpdateMax=1e220` is recommended for `Digits=200`.

Some features of Sentaurus Device do not take advantage of extended precision, including:

- Input and output to files
- Compact circuit models (see the *Compact Models User Guide*)
- CMI and the standard C++ interface in the PMI (see Part III of this user guide)
- Monte Carlo (see the *Sentaurus™ Device Monte Carlo User Guide*)
- Optical simulations
- Integration of beam distribution for optical generation (see the keyword `RecBoxIntegr` in [Transfer Matrix Method on page 669](#))

---

## System Command

The `System` command allows UNIX commands to be executed during a Sentaurus Device simulation:

```
System ( "UNIX command" )
```

The `System` command can appear as an independent command in the `Solve` section, as well as within a `Transient`, `Plugin`, or `Quasistationary` statement. The string argument of the `System` command is passed to a UNIX shell for evaluation.

By default, the return status of the UNIX command is ignored. If the following variant is used, Sentaurus Device examines the return status:

```
+System ( "UNIX command" )
```

The `System` command is considered to have converged if the return status is zero. Otherwise, it has not converged.

## References

- [1] R. E. Bank and D. J. Rose, “Global Approximate Newton Methods,” *Numerische Mathematik*, vol. 37, no. 2, pp. 279–295, 1981.

## **Part II: Physics in Sentaurus Device**

---

This part of the *Sentaurus™ Device User Guide* contains the following chapters:

- [Chapter 7, Poisson Equation and Quasi-Fermi Potentials](#)
- [Chapter 8, Carrier Transport Equations in Semiconductors](#)
- [Chapter 9, Lattice and Carrier Temperature Equations](#)
- [Chapter 10, Boundary Conditions](#)
- [Chapter 11, Transport Equations in Metals, Organic Materials, and Disordered Media](#)
- [Chapter 12, Semiconductor Band Structure](#)
- [Chapter 13, Incomplete Ionization](#)
- [Chapter 14, Quantization](#)
- [Chapter 15, Mobility](#)
- [Chapter 16, Generation–Recombination](#)
- [Chapter 17, Traps and Fixed Charges](#)
- [Chapter 18, Phase and State Transitions](#)
- [Chapter 19, Degradation](#)
- [Chapter 20, Organic Devices](#)
- [Chapter 21, Optical Generation](#)
- [Chapter 22, Radiation](#)
- [Chapter 23, Noise, Fluctuations, and Sensitivity](#)
- [Chapter 24, Tunneling](#)
- [Chapter 25, Hot-Carrier Injection](#)
- [Chapter 26, Heterostructure Device Simulation](#)
- [Chapter 27, Energy-Dependent Parameters](#)
- [Chapter 28, Anisotropic Properties](#)
- [Chapter 29, Ferroelectric Materials](#)

[Chapter 30, Ferromagnetism and Spin Transport](#)

[Chapter 31, Mechanical Stress](#)

[Chapter 32, Galvanic Transport](#)

[Chapter 33, Thermal Properties](#)

[Chapter 34, Light-Emitting Diodes](#)

[Chapter 35, Quantum Wells](#)

[Chapter 36, Kinetic Monte Carlo MIM Transport](#)

[Chapter 37, Kinetic Monte Carlo ReRAM](#)

# 7

## Poisson Equation and Quasi-Fermi Potentials

---

*This chapter discusses the Poisson equation, electrostatic potential, and quasi-Fermi potentials.*

In all semiconductor devices, mobile charges (electrons and holes) and immobile charges (ionized dopants or traps) play a central role. The charges determine the electrostatic potential and, in turn, are themselves affected by the electrostatic potential. Therefore, each electrical device simulation at the very least must compute the electrostatic potential.

When all contacts of a device are biased to the same voltage, the device is in equilibrium, and the electron and hole densities are described by a constant quasi-Fermi potential. Therefore, together with the electrostatic potential, the relation between the quasi-Fermi potentials and the electron and hole densities is sufficient to perform the simplest possible device simulation.

Maxwell–Boltzmann statistics is the default in Sentaurus Device, and you can choose to use Fermi–Dirac statistics as described in [Fermi Statistics on page 233](#).

---

### Electrostatic Potential

The electrostatic potential is the solution of the Poisson equation, which is:

$$\nabla \cdot (\epsilon \nabla \phi - \vec{P}) = -q(p - n + N_D - N_A) - \rho_{\text{trap}} \quad (37)$$

where:

- $\epsilon$  is the electrical permittivity.
- $\vec{P}$  is the ferroelectric polarization (see [Chapter 29 on page 905](#)).
- $q$  is the elementary electronic charge.
- $n$  and  $p$  are the electron and hole densities.
- $N_D$  is the concentration of ionized donors.

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

### Electrostatic Potential

- $N_A$  is the concentration of ionized acceptors.
- $\rho_{\text{trap}}$  is the charge density contributed by traps and fixed charges (see [Chapter 17 on page 543](#)).

The dataset name for the electrostatic potential is `ElectrostaticPotential`. The right-hand side of [Equation 37](#) (divided by  $-q$ ) is stored in the `SpaceCharge` dataset. The keyword for the Poisson equation in the `Solve` section is `Poisson`.

The dimensionless relative permittivity (that is, the permittivity in units of the vacuum permittivity  $\epsilon_0$ ) is given by the parameter `epsilon` in the `Epsilon` parameter set. The boundary conditions for the Poisson equation are discussed in [Chapter 10 on page 258](#).

---

## Dipole Layer

At material interfaces, dipole layers of immobile charges can occur, leading to a potential jump across the interface. They are modeled by:

$$\phi_2 - \phi_1 = \frac{q\sigma_D}{\epsilon_r\epsilon_0} \quad (38)$$

where:

- $\phi_1$  and  $\phi_2$  refer to the electrostatic potential at both sides of the interface (index 1 is the reference side).
- $\sigma_D$  is the dipole surface density.
- $\epsilon_0$  is the free space permittivity.
- $\epsilon_r$  is the relative interface permittivity.
- $q$  is the elementary electronic charge.

The dipole interface model can be invoked for insulator–insulator interfaces by specifying the following in the `Physics` section of the interface:

```
Dipole ( Reference = "R1" )
```

Here, `Reference` denotes the reference side of the interface, and it can be either a region name or the material name.

The parameters of the model are given in the `Dipole` parameter set of the region or material interface section in the parameter file, and are listed in [Table 27](#).

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

### Quasi-Fermi Potential With Boltzmann Statistics

Table 27     *Dipole model parameters*

Symbol	Parameter name	Default value	Unit
$\sigma_D$	sigma_D	0	$\text{cm}^{-1}$
$\epsilon_r$	epsilon	3.9	1

**Note:**

At critical points where heterointerfaces and dipole interfaces intersect, semiconductor regions must be interface connected. Interface traps with tunneling cannot be located on dipole interfaces.

---

## Equilibrium Solution

For specific models (see [Modified Ohmic Contacts on page 259](#), [Barrier Lowering at Schottky Contacts on page 265](#), [Floating Semiconductor Contacts on page 281](#), and [Conductive Insulators on page 300](#)), the equilibrium electrostatic potential is required internally. In this case, the Poisson equation is solved with equilibrium boundary conditions before any other `Coupled` statement and the solution is saved for use during the simulation.

You can control the parameters of the nonlinear solver for the equilibrium Poisson equation. The parameters `Iterations`, `Digits`, `NotDamped`, `LineSearchDamping`, and `RelErrControl`, described in [Table 340 on page 1719](#), can be specified as options to the keyword `EquilibriumSolution` in the `Math` section (see [Table 217 on page 1605](#)) to control the Newton solver behavior.

The following example forces the Newton solver to solve for an equilibrium solution with 7 digits as the target accuracy and a maximum number of 100 iterations:

```
Math {  
    ...  
    EquilibriumSolution(Iterations=100 Digits=7)  
    ...  
}
```

---

## Quasi-Fermi Potential With Boltzmann Statistics

Electron and hole densities can be computed from the electron and hole quasi-Fermi potentials, and vice versa.

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

### Quasi-Fermi Potential With Boltzmann Statistics

If Boltzmann statistics is assumed, the formulas read:

$$n = N_C \exp \left[ \frac{E_{F,n} - E_C}{kT} \right] \quad (39)$$

$$p = N_V \exp \left[ \frac{E_V - E_{F,p}}{kT} \right] \quad (40)$$

where:

- $N_C$  and  $N_V$  are the effective density-of-states.
- $E_{F,n} = -q\Phi_n$  and  $E_{F,p} = -q\Phi_p$  are the quasi-Fermi energies for electrons and holes.
- $\Phi_n$  and  $\Phi_p$  are electron and hole quasi-Fermi potentials, respectively.
- $E_C$  and  $E_V$  are conduction and valence band edges, defined as:

$$E_C = -\chi - q(\phi - \phi_{ref}) \quad (41)$$

$$E_V = -\chi - E_{g,eff} - q(\phi - \phi_{ref}) \quad (42)$$

where  $\chi$  denotes the electron affinity,  $E_{g,eff}$  is the effective band gap, and  $\phi_{ref}$  is a constant reference potential (see below). The zero level for the quasi-Fermi potential agrees with the zero level for the voltages applied at the contacts.

In unipolar devices, such as MOSFETs, it is sometimes possible to assume that the value of quasi-Fermi potential for the minority carrier is constant in certain regions. In this case, the concentration of the minority carrier can be directly computed from [Equation 39](#) or [Equation 40](#). This strategy is applied in Sentaurus Device if one of the carriers (electron or hole) is not specified inside the Coupled statement of the `Solve` section. Sentaurus Device uses an approximation scheme to determine the constant value of the quasi-Fermi potential.

#### Note:

In many cases if avalanche generation is important, the one carrier approximation cannot be applied even for unipolar devices.

The datasets for  $E_{F,n}$ ,  $\Phi_n$ ,  $E_{F,p}$ , and  $\Phi_p$  are named `eQuasiFermiEnergy`, `eQuasiFermiPotential`, `hQuasiFermiEnergy`, and `hQuasiFermiPotential`, respectively.

The `ConstRefPot` parameter allows you to specify  $\phi_{ref}$  explicitly. Otherwise, Sentaurus Device computes  $\phi_{ref}$  from the vacuum level, using the following rules:

- If there is silicon in any simulated semiconductor structure, then the intrinsic Fermi level of silicon is selected as reference,  $\phi_{ref} = \Phi_{intr}(Si)$ .
- Otherwise, if any simulated device structure contains GaAs, then  $\phi_{ref} = \Phi_{intr}(GaAs)$ .
- In all other cases, Sentaurus Device selects the material with the smallest band gap (assuming a mole fraction of 0) and takes the value of its intrinsic Fermi level as  $\phi_{ref}$ .

## Fermi Statistics

For the equations presented in the previous section, Boltzmann statistics was assumed for electrons and holes. Physically more correct, Fermi (also called Fermi–Dirac) statistics can be used. Fermi statistics becomes important for high values of carrier densities, for example,  $n > 1 \times 10^{19} \text{ cm}^{-3}$  in the active regions of a silicon device.

For Fermi statistics, [Equation 39](#) and for [Equation 40](#) are replaced by:

$$n = N_C F_{1/2} \frac{e^{E_{F,n} - E_C}}{kT} \quad (43)$$

$$p = N_V F_{1/2} \frac{e^{E_V - E_{F,p}}}{kT} \quad (44)$$

where  $F_{1/2}$  is the Fermi integral of order 1/2.

Alternatively, you can write these expressions as:

$$n = \gamma_n N_C \exp \left( \frac{E_{F,n} - E_C}{kT} \right) \quad (45)$$

$$p = \gamma_p N_V \exp \left( \frac{E_V - E_{F,p}}{kT} \right) \quad (46)$$

where  $\gamma_n$  and  $\gamma_p$  are the functions of  $\eta_n$  and  $\eta_p$ :

$$\gamma_n = \frac{n}{N_C} \exp(-\eta_n) \quad (47)$$

$$\gamma_p = \frac{p}{N_V} \exp(-\eta_p) \quad (48)$$

$$\eta_n = \frac{E_{F,n} - E_C}{kT} \quad (49)$$

$$\eta_p = \frac{E_V - E_{F,p}}{kT} \quad (50)$$


---

## Using Fermi Statistics

To activate Fermi statistics, you must specify the keyword `Fermi` in the `Physics` section:

```
Physics {
    ...
    Fermi
}
```

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

Initial Guess for Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells

### Note:

Fermi statistics can be activated only for the entire device. Region-specific or material-specific activation is not possible, and the keyword `Fermi` can be defined in any `Physics` section.

---

## Initial Guess for Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells

To determine an initial guess for the electrostatic potential and quasi-Fermi potentials, a device is divided into doping well regions, where a doping well region is a semiconductor region consisting of a set of connected semiconductor elements bounded by nonsemiconductor elements or by vacuum (a doping well region might contain several mesh semiconductor regions). Then, each doping well region is further subdivided into wells of n-type and p-type doping, such that p-n junctions (`DopingConcentration=0`) serve as dividers between wells. For doping wells with more than one contact, the wells are further subdivided, such that no well is associated with more than one contact. Every well is connected uniquely to a contact or it has no contact (floating well).

### Note:

Sentaurus Device uses integers to enumerate all doping wells. You can visualize the indices of the doping wells by plotting the `DopingWells` field.

In wells with contacts, the quasi-Fermi potential of the majority carrier is set to the voltage on the contact associated with the well. For wells that have no contacts, the following equations define the quasi-Fermi potential for the majority carriers:

$$\Phi_p = k_{\text{float}} V_{\max} + (1 - k_{\text{float}}) V_{\min} \quad (51)$$

$$\Phi_n = (1 - k_{\text{float}}) V_{\max} + k_{\text{float}} V_{\min} \quad (52)$$

where:

- $V_{\min}$  and  $V_{\max}$  are the minimum and maximum of the contact voltages of the semiconductor wells with contacts in the doping well region, where the contactless well under consideration is located.
- The coefficient  $k_{\text{float}}$  is an adjustable parameter with a default value of 0. To change the value of  $k_{\text{float}}$ , use the keyword `FloatCoef` in the `Physics` section.

If a well has a contact and it is the only well in the doping well region to which it belongs, then the quasi-Fermi potential of the minority carrier is set equal to the quasi-Fermi potential of the majority carrier. For all other wells, the quasi-Fermi potential of the minority carrier is set to  $V_{\min}$  or  $V_{\max}$  if the well is n-type or p-type, respectively, with  $V_{\min}$  and  $V_{\max}$  previously described.

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

Alternative Initial Guess for Quasi-Fermi Potentials From Linear Conduction Equation

The electrostatic potential in a well is set to the value of the quasi-Fermi potential of the majority carrier adjusted by the built-in potential, that is, the initial solution will obey charge neutrality in all semiconductor regions.

---

### Regionwise Specification of Initial Quasi-Fermi Potentials

You can set initial quasi-Fermi potentials regionwise. This is especially useful in charge-coupled device (CCD) simulations, where a CCD cell or region must be initially in a certain state (usually, deep depletion). By specifying an initial quasi-Fermi potential, the region or set of regions can be brought to the proper state or states at the start of the simulation.

The initial quasi-Fermi potentials for electrons and holes can be specified regionwise in the `Physics` section using `eQuasiFermi` for electrons and `hQuasiFermi` for holes, followed by the value in volts:

```
Physics(Region="region_1") {  
    eQuasiFermi = 10  
}
```

The initial quasi-Fermi potential is recomputed when the continuity equation for the respective carrier is solved. If the continuity equation for the carrier whose quasi-Fermi potential has been specified is not solved, then the quasi-Fermi potential does not change. In this case, the device can be biased to an initial state through a quasistationary or transient simulation, while keeping the initial specified quasi-Fermi potential.

---

### Alternative Initial Guess for Quasi-Fermi Potentials From Linear Conduction Equation

Sentaurus Device provides an alternative method to initialize the quasi-Fermi potentials in semiconductor materials. It is more computationally intensive than their initialization in doping wells, but often gives better results, especially in complicated device structures with multiple contacts per doping well region.

In this approach, a solution of the linear conduction equations:

$$-\nabla \cdot (\sigma_{n_0} \nabla \Phi_n) = 0 \quad (53)$$

$$-\nabla \cdot (\sigma_{p_0} \nabla \Phi_p) = 0 \quad (54)$$

serves as an initial guess for the electron and hole quasi-Fermi potentials,  $\Phi_n$  and  $\Phi_p$ , respectively.

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

Alternative Initial Guess for Quasi-Fermi Potentials From Linear Conduction Equation

The conductivity coefficients for electrons ( $\sigma_{n_0}$ ) and holes ( $\sigma_{p_0}$ ) are approximated as:

$$\sigma_{n_0} = q\mu_{L,n}n_0 \quad (55)$$

$$\sigma_{p_0} = q\mu_{L,p}p_0 \quad (56)$$

where  $n_0$ ,  $p_0$  are the equilibrium carrier concentrations, and  $\mu_{L,n}$ ,  $\mu_{L,p}$  are the constant carrier mobilities. The concentrations  $n_0$  and  $p_0$  are calculated by invoking the charge neutrality condition ([Equation 99 on page 258](#)) and the law of mass action ([Equation 100 on page 258](#)). The mobilities  $\mu_{L,n}$  and  $\mu_{L,p}$  are taken as the values of the `mumax` parameter for electrons and holes, respectively, from the `ConstantMobility` parameter set (see [Mobility due to Phonon Scattering on page 386](#)).

The following boundary conditions are used when solving [Equation 53](#) and [Equation 54](#):

- At contacts, the Dirichlet boundary condition is applied, by setting the quasi-Fermi potentials equal to the voltage on the contact.
- For interfaces between a semiconductor and an insulator, a homogeneous Neumann boundary condition is imposed.
- At semiconductor–semiconductor and semiconductor–metal interfaces, the continuity of quasi-Fermi potentials is enforced.

In the command file, the initialization of quasi-Fermi potentials from the solution of [Equation 53](#) and [Equation 54](#) is activated by the `IniteQuasiFermi` and `InithQuasiFermi` options, respectively, set in the `Physics` section. You can use these options independently and can also use them in regionwise or materialwise `Physics` sections. The following example activates the linear conduction equation for the electron quasi-Fermi potential in region `n-epi`:

```
Physics (Region="n-epi") {
    IniteQuasiFermi
}
```

For interior interfaces where the linear conduction equation is solved only in one of the regions of the interface, the continuity condition for quasi-Fermi potentials is evaluated based on the initial guess coming from doping wells (see [Initial Guess for Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells on page 234](#)). In metal regions, [Equation 53](#) and [Equation 54](#) are never solved. Instead, the metal conductivity equation is used to compute the Fermi level (see [Transport in Metals on page 295](#)).

Therefore, the linear conduction equation is solved by default only in semiconductor regions. However, the solution of the linear conduction equation can be extended to insulator regions by using the `includingInsulator` option. For example:

```
IniteQuasiFermi(includingInsulator)
InithQuasiFermi(includingInsulator)
```

## Chapter 7: Poisson Equation and Quasi-Fermi Potentials

### Electrode Charge Calculation

In insulator regions, the conductivity coefficients and  $\sigma_{n_0}$  and  $\sigma_{p_0}$ , in [Equation 53](#) and [Equation 54](#), are approximated as  $\sigma_{n_0} = \sigma_{p_0} = 1/\rho_0$ , where  $\rho_0$  is the resistivity value of the corresponding insulator material. The resistivity value  $\rho_0$  is taken as the value of the Resist0 parameter from the Resistivity parameter set (see [Transport in Metals on page 295](#)).

If the continuity equation for the carrier whose quasi-Fermi potential has been specified is not solved, then the initial quasi-Fermi potential is recomputed for every ramping step. In this case, the concentration of the carrier is calculated directly from [Equation 39](#) or [Equation 40](#) (see [Quasi-Fermi Potential With Boltzmann Statistics on page 231](#)).

You can access the datasets storing the values of  $\Phi_n$ ,  $\sigma_{n_0}$ ,  $\Phi_p$ , and  $\sigma_{p_0}$  by using the keywords eQuasiFermiPotential, eInitialConductivity/Element, hQuasiFermiPotential, and eInitialConductivity/Element, respectively.

---

## Electrode Charge Calculation

Sentaurus Device outputs the electrode charge to the current plot file (\*.plt file) after each bias or time point.

For an electrode that contacts an insulator region only, the charge is computed from Gauss's law and represents the charge that would sit on the surface of a real contact to the device.

For an electrode that contacts a semiconductor region, the charge also is computed from Gauss's law; however, the Gaussian surface used for the integration includes the doping well associated with the electrode. In this case, the electrode charge represents the charge that sits on the electrode plus the space charge in the doping well.

# 8

## Carrier Transport Equations in Semiconductors

---

This chapter discusses carrier transport models for electrons and holes in inorganic semiconductors.

---

### Introduction to Carrier Transport Models

Sentaurus Device supports several carrier transport models for semiconductors. They all can be written in the form of continuity equations, which describe charge conservation:

$$\nabla \cdot \vec{J}_n = q(R_{\text{net},n} - G_{\text{net},n}) + q \frac{\partial n}{\partial t} \quad -\nabla \cdot \vec{J}_p = q(R_{\text{net},p} - G_{\text{net},p}) + q \frac{\partial p}{\partial t} \quad (57)$$

where:

- $R_{\text{net},n}$  and  $R_{\text{net},p}$  are the electron and hole net recombination rate, respectively.
- $G_{\text{net},n}$  and  $G_{\text{net},p}$  are the electron and hole net generation rate, respectively.
- $\vec{J}_n$  is the electron current density.
- $\vec{J}_p$  is the hole current density.
- $n$  and  $p$  are the electron and hole density, respectively.

The transport models differ in the expressions used to compute  $\vec{J}_n$  and  $\vec{J}_p$ . These expressions are the topic of this chapter. Additional equations to compute temperatures are usually also considered part of a transport model but are deferred to [Chapter 9 on page 245](#). Models for  $R_{\text{net},n}$  and  $R_{\text{net},p}$  are discussed in [Chapter 16 on page 473](#), [Chapter 17 on page 543](#), and [Chapter 24 on page 819](#). Boundary conditions for Equation 57 are discussed in [Chapter 10 on page 258](#).

## Chapter 8: Carrier Transport Equations in Semiconductors

### Drift-Diffusion Model

Depending on the device under investigation and the level of modeling accuracy required, you can select different transport models:

- **Drift-diffusion:** Isothermal simulation, suitable for low-power density devices with long active regions (see [Drift-Diffusion Model on page 239](#)).
- **Thermodynamic:** Accounts for self-heating. Suitable for devices with low thermal exchange, particularly, high-power density devices with long active regions (see [Thermodynamic Model for Current Densities on page 240](#)).
- **Hydrodynamic:** Accounts for energy transport of the carriers. Suitable for devices with small active regions (see [Hydrodynamic Model for Current Densities on page 241](#)).
- **Monte Carlo:** Solves the Boltzmann equation for a full band structure.

The numeric approach for the Monte Carlo method (and the physical models usable with it) differs significantly from the other transport models. Therefore, the Monte Carlo method is described separately (see the *Sentaurus™ Device Monte Carlo User Guide*).

For all other transport models, the solution of [Equation 57](#) is requested by the keywords `Electron` and `Hole` in the `Solve` section. When the equation for a density is not solved in a `Solve` statement, the quasi-Fermi potential for the respective carrier type remains fixed, unless the keyword `RecomputeQFP` is present in the `Math` section and the equation for the other density is solved.

Solutions of the densities  $n$  and  $p$  are stored in the datasets `eDensity` and `hDensity`, respectively. Current densities  $J_n$  and  $J_p$  are stored in the vector datasets `eCurrentDensity` and `hCurrentDensity`, their sum is stored in `ConductionCurrentDensity`, and the total current density (including displacement current) is stored in `TotalCurrentDensity`. For an alternative representation of the total current density, see [Current Potential on page 242](#).

#### Note:

Maxwell–Boltzmann statistics is the default for transport models in Sentaurus Device, and you can choose to use Fermi–Dirac statistics as described in [Fermi Statistics on page 233](#).

---

## Drift-Diffusion Model

The drift-diffusion model is the default carrier transport model in Sentaurus Device. For the drift-diffusion model, the current densities for electrons and holes are given by:

$$\vec{J}_n = \mu_n(n\nabla E_C - 1.5nkT\nabla \ln m_n) + D_n(\nabla n - n\nabla \ln \gamma_n) \quad (58)$$

$$\vec{J}_p = \mu_p(p\nabla E_V + 1.5pkT\nabla \ln m_p) - D_p(\nabla p - p\nabla \ln \gamma_p) \quad (59)$$

## Chapter 8: Carrier Transport Equations in Semiconductors

### Thermodynamic Model for Current Densities

The first term takes into account the contribution due to the spatial variations of the electrostatic potential, the electron affinity, and the band gap. The remaining terms take into account the contribution due to the gradient of concentration, and the spatial variation of the effective masses  $m_n$  and  $m_p$ . For Fermi statistics,  $\gamma_n$  and  $\gamma_p$  are given by [Equation 47](#) and [Equation 48](#). For Boltzmann statistics,  $\gamma_n = \gamma_p = 1$ .

By default, the diffusivities  $D_n$  and  $D_p$  are given through the mobilities by the Einstein relation,  $D_n = kT\mu_n$  and  $D_p = kT\mu_p$ . However, you can compute them independently (see [Non-Einstein Diffusivity on page 453](#)).

When the Einstein relation holds, the current equations can be simplified to:

$$\vec{J}_n = -nq\mu_n \nabla \Phi_n \quad (60)$$

$$\vec{J}_p = -pq\mu_p \nabla \Phi_p \quad (61)$$

where  $\Phi_n$  and  $\Phi_p$  are the electron and hole quasi-Fermi potentials, respectively (see [Quasi-Fermi Potential With Boltzmann Statistics on page 231](#)).

You can use the drift-diffusion model with a lattice temperature equation, but it is not mandatory. It is not possible to use the drift-diffusion model for a particular carrier type and to solve the carrier temperature for the same carrier type; the hydrodynamic model is required for that (see [Hydrodynamic Model for Current Densities on page 241](#)).

---

## Thermodynamic Model for Current Densities

In the thermodynamic model [1], the relations [Equation 60](#) and [Equation 61](#) are generalized to include the temperature gradient as a driving term:

$$\vec{J}_n = -nq\mu_n (\nabla \Phi_n + P_n \nabla T) \quad (62)$$

$$\vec{J}_p = -pq\mu_p (\nabla \Phi_p + P_p \nabla T) \quad (63)$$

where  $P_n$  and  $P_p$  are the absolute thermoelectric powers [2] (see [Thermoelectric Power on page 1032](#)), and  $T$  is the lattice temperature.

The model differs from drift-diffusion when the lattice temperature equation is solved (see [Thermodynamic Model for Lattice Temperature on page 249](#)). However, it is possible to solve the lattice temperature equation even when using the drift-diffusion model.

To activate the thermodynamic model, specify the `Thermodynamic` keyword in the global `Physics` section.

## Hydrodynamic Model for Current Densities

In the hydrodynamic model, current densities are defined as:

$$\vec{J}_n = \mu_n n \nabla E_C + kT_n \nabla n - nkT_n \nabla \ln \gamma_n + \lambda_n f_n^{\text{td}} kn \nabla T_n - 1.5nkT_n \nabla \ln m_n \quad (64)$$

$$\vec{J}_p = \mu_p (p \nabla E_V - kT_p \nabla p + pkT_p \nabla \ln \gamma_p - \lambda_p f_p^{\text{td}} kp \nabla T_p + 1.5pkT_p \nabla \ln m_p) \quad (65)$$

The first term takes into account the contribution due to the spatial variations of electrostatic potential, electron affinity, and the band gap. The remaining terms in [Equation 64](#) and [Equation 65](#) take into account the contribution due to the gradient of concentration, the carrier temperature gradients, and the spatial variation of the effective masses  $m_n$  and  $m_p$ . For Fermi statistics,  $\gamma_n$  and  $\gamma_p$  are given by [Equation 47](#) and [Equation 48](#), and  $\lambda_n = F_{1/2}(\eta_n)/F_{-1/2}(\eta_n)$  and  $\lambda_p = F_{1/2}(\eta_p)/F_{-1/2}(\eta_p)$  (with  $\eta_n$  and  $\eta_p$  from [Equation 49](#) and [Equation 50](#)); for Boltzmann statistics,  $\gamma_n = \gamma_p = \lambda_n = \lambda_p = 1$ .

The thermal diffusion constants  $f_n^{\text{td}}$  and  $f_p^{\text{td}}$  are available in the `ThermalDiffusion` parameter set. They default to zero, which corresponds to the Stratton model [3][4].

[Equation 64](#) and [Equation 65](#) assume that the Einstein relation  $D = \mu kT$  holds ( $D$  is the diffusion constant), which is true only near the equilibrium. Therefore, Sentaurus Device provides the option to replace the carrier temperatures in [Equation 64](#) and [Equation 65](#) by  $gT_c + (1-g)T$ , an average of the carrier temperature ( $T_n$  or  $T_p$ ) and the lattice temperature. This can be useful in devices where the carrier diffusion is important. The coefficient  $g$  for electrons and holes can be specified in the `ThermalDiffusion` parameter set. It defaults to one.

To activate the hydrodynamic model, specify the keyword `Hydrodynamic` in the global `Physics` section. If only one carrier temperature equation is to be solved, then `Hydrodynamic` must be specified with an option, either `Hydrodynamic(eTemperature)` or `Hydrodynamic(hTemperature)`.

---

## Numeric Parameters for Continuity Equation

Carrier concentrations must never be negative. If during a Newton iteration a concentration erroneously becomes negative, then Sentaurus Device applies damping procedures to make it positive. The concentration that finally results from a Newton iteration is limited to a value that can be specified in  $\text{cm}^{-3}$  by `DensLowLimit=<\text{float}>` in the global `Math` section (default:  $10^{-100} \text{ cm}^{-3}$ ).

### Note:

For very-low device temperatures (below 50 K), the carrier concentration might be extremely small. To support simulations using such temperatures, specify `ExtendedPrecision` (see [Extended Precision on page 223](#)) and

## Chapter 8: Carrier Transport Equations in Semiconductors

### Numeric Approaches for Contact Current Computation

DensLowLimitPower=<int> in the global `Math` section (default: -100). The lowest carrier concentration limit will be set to the minimum between DensLowLimit=<float> and  $10^{\text{DensLowLimitPower}}$  cm<sup>-3</sup>.

---

## Numeric Approaches for Contact Current Computation

By default, Sentaurus Device computes the contact current as the sum of the integral of the current density over the surface of the doping well associated with the contact and the integral of the charge generation rate over the volume of the doping well. The following alternative, mutually exclusive approaches can be selected in the global `Math` section:

- `CurrentWeighting` activates a domain-integration method that uses a solution-dependent weighting function to minimize numeric errors (see [5] for details).
- `DirectCurrent` activates computation of the current as the surface integral of the current density over the contact area.

**Note:**

The current that flows into circuit nodes in mixed-mode simulations is always computed with the `DirectCurrent` approach, regardless of the specification in the `Math` section.

---

## Current Potential

The total current density:

$$\vec{J} = \vec{J}_n + \vec{J}_p + \vec{J}_D \quad (66)$$

satisfies the conservation law  $\nabla \cdot \vec{J} = 0$ .

Consequently,  $\vec{J}$  can be written as the curl of a vector potential  $\vec{W}$ :

$$\vec{J} = \nabla \times \vec{W} \quad (67)$$

In a 2D simulation,  $\vec{W}$  only has a nonzero component along the z-axis, which is written simply as  $W_z = W$ .

The total current density is then given by:

$$\vec{J} = \begin{bmatrix} \frac{\partial W}{\partial y} \\ -\frac{\partial W}{\partial x} \end{bmatrix} \quad (68)$$

The function  $W$  has the following important properties:

- The contour lines of  $W$  are the current lines of  $\vec{J}$ .
- The difference between the values of  $W$  at any two points equals the total current flowing across any line linking these points.

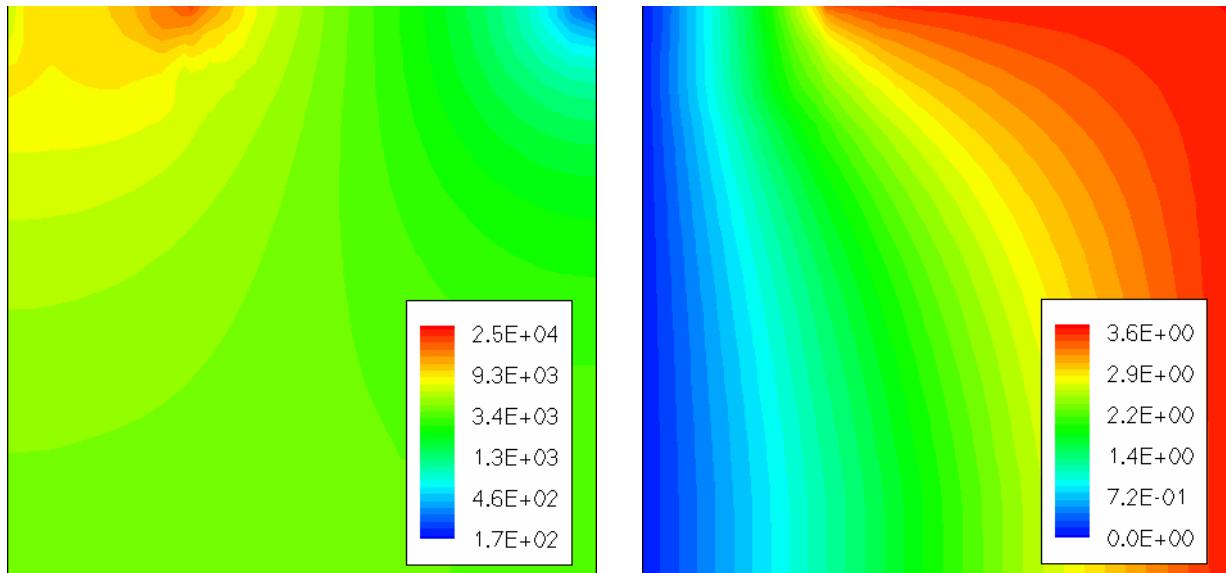
Sentaurus Device computes the 2D current potential according to the approach proposed by Palm and Van de Wiele [6].

To visualize the results, add the keyword `CurrentPotential` to the `Plot` section:

```
Plot { CurrentPotential }
```

[Figure 18](#) displays plots of the total current density and current potential for a simple square device.

Figure 18 (Left) Total current density [ $A\text{cm}^{-2}$ ] and (right) current potential [ $A\text{cm}^{-1}$ ]



## References

- [1] G. Wachutka, "An Extended Thermodynamic Model for the Simultaneous Simulation of the Thermal and Electrical Behaviour of Semiconductor Devices," in *Proceedings of the Sixth International Conference on the Numerical Analysis of Semiconductor Devices and Integrated Circuits (NASECODE VI)*, Dublin, Ireland, pp. 409–414, July 1989.
- [2] H. B. Callen, *Thermodynamics and an Introduction to Thermostatistics*, New York: John Wiley & Sons, 2nd ed., 1985.

## Chapter 8: Carrier Transport Equations in Semiconductors

### References

- [3] R. Stratton, "Diffusion of Hot and Cold Electrons in Semiconductor Barriers," *Physical Review*, vol. 126, no. 6, pp. 2002–2014, 1962.
- [4] Y. Apanovich *et al.*, "Numerical Simulation of Submicrometer Devices Including Coupled Nonlocal Transport and Nonisothermal Effects," *IEEE Transactions on Electron Devices*, vol. 42, no. 5, pp. 890–898, 1995.
- [5] P. D. Yoder *et al.*, "Optimized Terminal Current Calculation for Monte Carlo Device Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 10, pp. 1082–1087, 1997.
- [6] E. Palm and F. Van de Wiele, "Current Lines and Accurate Contact Current Evaluation in 2-D Numerical Simulation of Semiconductor Devices," *IEEE Transactions on Electron Devices*, vol. ED-32, no. 10, pp. 2052–2059, 1985.

# 9

## Lattice and Carrier Temperature Equations

---

*This chapter describes the equations for lattice and carrier temperatures.*

---

### Introduction to Temperature Equations

Sentaurus Device can compute up to three different temperatures: lattice temperature, electron temperature, and hole temperature. Lattice temperature describes self-heating of devices. The electron and hole temperatures are required to model nonequilibrium effects in deep-submicron devices (in particular, velocity overshoot and impact ionization).

The following options are available:

- The lattice temperature can be computed from the total dissipated heat, assuming the temperature is constant throughout the device.
- The lattice temperature can be computed nonuniformly using the default model, or the thermodynamic model, or the hydrodynamic model.
- One or both carrier temperatures can be computed using the hydrodynamic model.

Irrespective of the model used, the lattice temperature is stored in the dataset `Temperature`, and electron and hole temperatures are stored as `eTemperature` and `hTemperature`, respectively.

To solve the lattice temperature, you must specify a thermal boundary condition (see [Thermal Boundary Conditions on page 283](#)). To solve the carrier temperatures, no thermal boundary conditions are required.

By default (and as an initial guess), the lattice temperature is set to 300 K throughout the device or to the value of `Temperature` set in the `Physics` section. If the device has one or more defined thermodes, an average temperature is calculated from the temperatures at the thermodes and is set throughout the device. As an initial guess, electron and hole temperatures are set to the lattice temperature. Carrier temperatures that are not solved are assumed to equal the lattice temperature throughout the simulation.

## Uniform Self-Heating

A simple self-heating model is available to account for self-heating effects without solving the lattice heat flow equation. The self-heating effect can be captured with a uniform temperature, which is bias dependent or current dependent. The global temperature is computed from a global heat balance equation where the dissipated power  $P_{\text{diss}}$  is equal to the sum of the boundary heat fluxes (on thermodes with finite thermal resistance):

$$P_{\text{diss}} = \sum_i \frac{T - T_{\text{thermode}}^{(i)}}{R_{\text{th}}^{(i)}} + \sum_k \frac{T - T_{\text{circ}}^{(k)}}{R_{\text{th}}^{(k)}} \quad (69)$$

where:

- $T$  is the device global temperature.
- $T_{\text{thermode}}^{(i)}$  and  $R_{\text{th}}^{(i)}$  are the temperature and thermal resistivity of thermode  $i$  not connected in a thermal circuit.
- $T_{\text{circ}}^{(k)}$  and  $R_{\text{th}}^{(k)}$  are the temperature and thermal resistivity of thermode  $k$  connected in a thermal circuit.

The second sum in Equation 69 becomes zero when the device is not connected to any thermal circuit.

For a thermode  $k$  connected in a thermal circuit, an additional thermal circuit equation is solved for the temperature at the respective thermode  $T_{\text{circ}}^{(k)}$ . The thermal circuit equation solved is derived from the condition that the heat flux through the connected thermode is equal to the flux flowing through the thermal circuit element to which the thermode is connected.

In the case of thermode  $k$  connected to a thermal resistor, the thermal circuit equation becomes:

$$(T - T_{\text{circ}}^{(k)})/R_{\text{th}}^{(k)} = \nabla T^{\text{resistor}}/R_{\text{th}} \quad (70)$$

where:

- The left-hand side represents the thermal flux at thermode  $k$ .
- The right-hand side represents the heat flux flowing through the thermal resistor with thermal resistivity  $R_{\text{th}}$ .
- $\nabla T^{\text{resistor}} = T_{\text{resistor}} - T_{\text{circ}}^{(k)}$  is the temperature drop across the thermal resistor.

For a thermal capacitor, the thermal equation becomes  $(T - T_{\text{circ}}^{(k)})/R_{\text{th}}^{(k)} = C_{\text{th}} \frac{d}{dt} \nabla T^{\text{capacitor}}$  where  $\nabla T^{\text{capacitor}} = T_{\text{capacitor}} - T_{\text{circ}}^{(k)}$  is the temperature drop across the thermal capacitor in a transient simulation.

## Chapter 9: Lattice and Carrier Temperature Equations

### Uniform Self-Heating

If a thermode does not specify a thermal resistance or a thermal conductance, then by default, a Dirichlet boundary condition of a fixed thermode temperature is assumed. In this case, the thermode is not accounted for in the sum on the right-hand side of [Equation 69](#) because the respective heat flux is zero. When none of the thermodes specifies any thermal resistance or conductance, [Equation 69](#) cannot be solved.

The dissipated power  $P_{\text{diss}}$  can be calculated as either the sum of all terminal currents multiplied by their respective terminal voltages ( $\sum IV$ ), the sum of  $IVs$  from the user-specified node, or the total Joule heat  $(J \cdot F)dv$ .

The global temperature  $T_i$  for the point  $t_i$  (in transient or quasistationary simulations) is computed based on the estimation of  $T_i$  at the previous point  $t_{i-1}$  and the solution temperature  $T_{i-1}$  at the same point  $t_{i-1}$ .

#### Note:

For uniform self-heating, the temperature is obtained by postprocessing, rather than computed self-consistently with all other solution variables. Therefore, the simulation results might depend on the simulation time step used in quasistationary or transient simulations. In addition, uniform self-heating must not be used for AC, noise, or harmonic balance simulations.

---

## Using Uniform Self-Heating

The uniform self-heating equation is activated in the global `Physics` section using the `PostTemperature` keyword:

```
Physics {
    PostTemperature
    ...
}
```

The heat fluxes at thermodes are defined in the `Thermode` sections by specifying `Temperature` and `SurfaceResistance` or `SurfaceConductance`:

```
Thermode {
    {Name= "top" Temperature=300 SurfaceConductance=0.1}
    ...
    {Name= "bottom" Temperature=300 SurfaceResistance=0.01}
}
```

The way  $P_{\text{diss}}$  is computed can be chosen by options to `PostTemperature` in the command file:

- As  $IV$  over all electrodes:

```
Physics {
    PostTemperature(IV_diss)
    ...
}
```

## Chapter 9: Lattice and Carrier Temperature Equations

### Default Model for Lattice Temperature

- As  $IV$  at user-selected electrodes:

```
Physics {
    PostTemperature(IV_diss("contact1" "contact2"))
    ...
}
```

- As the integral of Joule heat over the device volume:

```
Physics {
    PostTemperature
    ...
}
```

Uniform self-heating can be used during a quasistationary or transient simulation. As the feature replaces the lattice heat flow equation ([Equation 71](#), [Equation 72](#), and [Equation 78](#)), it cannot be used with the lattice temperature equation activated in the `Coupled` section.

---

## Default Model for Lattice Temperature

Sentaurus Device can compute a spatially dependent lattice temperature even when neither the thermodynamic nor the hydrodynamic model is activated. The equation for the temperature is:

$$\frac{\partial W_L}{\partial t} + \nabla \cdot \vec{S}_L = \frac{dW_L}{dt} \Big|_{coll} + \frac{1}{q} \vec{J}_n \cdot \nabla E_C + \frac{dW_n}{dt} \Big|_{coll} + \frac{1}{q} \vec{J}_p \cdot \nabla E_V + \frac{dW_p}{dt} \Big|_{coll} \quad (71)$$

(See [Hydrodynamic Model for Current Densities on page 241](#) for an explanation of the symbols.) That is, the model is similar to the hydrodynamic model, but all temperatures merge into the lattice temperature, and all heating terms are accounted for in the lattice temperature equation.

To use this model, specify the keyword `Temperature` in the `Solve` section. You do not have to specify a model in the `Physics` section.

To account for Peltier heat at a contact, or a metal–semiconductor interface, or a conductive insulator–semiconductor interface, switch on the Peltier heating model explicitly. See [Heating at Contacts, Metal–Semiconductor and Conductive Insulator–Semiconductor Interfaces on page 1035](#).

## Chapter 9: Lattice and Carrier Temperature Equations

### Thermodynamic Model for Lattice Temperature

---

## Thermodynamic Model for Lattice Temperature

With the thermodynamic model, the lattice temperature  $T$  is computed from:

$$\begin{aligned}\frac{\partial}{\partial t}(c_L T) - \nabla \cdot (\kappa \nabla T) &= -\nabla \cdot [(P_n T + \Phi_n) \vec{J}_n + (P_p T + \Phi_p) \vec{J}_p] \\ &\quad - \frac{1}{q} E_C + \frac{3}{2} kT (\nabla \cdot \vec{J}_n - q R_{\text{net},n}) \\ &\quad - \frac{1}{q} E_V + \frac{3}{2} kT (-\nabla \cdot \vec{J}_p - q R_{\text{net},p}) + \hbar \omega G^{\text{opt}}\end{aligned}\tag{72}$$

where:

- $\kappa$  is the thermal conductivity (see [Thermal Conductivity on page 1019](#)).
- $c_L$  is the lattice heat capacity (see [Heat Capacity on page 1017](#)).
- $E_C$  and  $E_V$  are the conduction and valence band energies, respectively.
- $G^{\text{opt}}$  is the optical generation rate from photons with frequency  $\omega$  (see [Chapter 21 on page 648](#)).
- $R_{\text{net},n}$  and  $R_{\text{net},p}$  are the electron and hole net recombination rates, respectively.
- Current densities  $J_n$  and  $J_p$  are computed as described in [Thermodynamic Model for Current Densities on page 240](#).

In metals, [Equation 72](#) degenerates into:

$$\frac{\partial}{\partial t}(c_L T) - \nabla \cdot (\kappa \nabla T) = -\nabla \cdot [(P T + \Phi_M) \vec{J}_M]\tag{73}$$

where:

- $P$  is the metal thermoelectric power.
- $\Phi_M$  is the Fermi potential in the metal.
- $\vec{J}_M$  is the metal current density as defined in [Equation 142](#).

---

## Total Heat and Its Contributions

The RHS of [Equation 72](#) is the total heat  $H$ . In the stationary case, the second term and the third term disappear, that is, the total heat is then given by:

$$H = -\nabla \cdot [(P_n T + \Phi_n) \vec{J}_n + (P_p T + \Phi_p) \vec{J}_p] + \hbar \omega G^{\text{opt}}\tag{74}$$

## Chapter 9: Lattice and Carrier Temperature Equations

### Thermodynamic Model for Lattice Temperature

The electron part can be rewritten as:

$$-\nabla \cdot [(P_n T + \Phi_n) \vec{J}_n] = \frac{1}{qn\mu_n} |\vec{J}_n|^2 - qT \nabla P_n \cdot \vec{J}_n + [(P_n T + \Phi_n) \vec{J}_n \cdot n_S]_S \delta_S - q(P_n T + \Phi_n) R_{\text{net},n} \quad (75)$$

where:

- $n_S$  is the surface normal at region interface  $S$ .
- $\delta_S$  denotes the surface delta function at region interface  $S$ .
- $[\alpha]_S$  denotes the jump of a function  $\alpha$  across the region interface  $S$ .

The resulting four terms are often denoted as the electron part of the Joule heat, Thomson heat, Peltier heat, and recombination heat, respectively.

---

## Using the Thermodynamic Model

To use the thermodynamic model, you must specify the keyword `Temperature` in the `Coupled` statement of the `Solve` section. Use the keyword `Thermodynamic` in the `Physics` section to activate extra terms in the current density equations (due to the gradient of the temperature).

The heating mechanisms of nonlocal line tunneling are not accounted for, by default, in the thermodynamic model. Therefore, to also take into account heating by nonlocal line tunneling, you must add the option `NLLTunnelingRecGenHeat` to the `Thermodynamic` keyword as `Thermodynamic(NLLTunnelingRecGenHeat)`.

To account for Peltier heat at a contact, or a metal–semiconductor interface, or a conductive insulator–semiconductor interface, switch on the Peltier heating model explicitly (see [Heating at Contacts, Metal–Semiconductor and Conductive Insulator–Semiconductor Interfaces on page 1035](#)).

Sentaurus Device allows the total heat generation rate to be plotted and the separate components of the total heat to be estimated and plotted. The total heat generation rate is the term on the right-hand side of [Equation 72](#). It is plotted using the `TotalHeat` keyword in the `Plot` section. The total heat is calculated only when Sentaurus Device solves the temperature equation.

[Table 28](#) shows the formulas to estimate individual heating mechanisms and the appropriate keywords to use in the `Plot` section of the command file (see [Device Plots on page 177](#)). The individual heat terms that are used for plotting and that are written to the `.log` file are less accurate than those Sentaurus Device uses to solve the transport equations. They serve as illustrations only.

## Chapter 9: Lattice and Carrier Temperature Equations

Thermodynamic Model for Lattice Temperature

Table 28 Terms and keywords used in Plot section of command file

Heat name	Keyword	Formula
Electron Joule heat	eJouleHeat	$\frac{ \vec{J}_n ^2}{qn\mu_n}$
Hole Joule heat	hJouleHeat	$\frac{ \vec{J}_p ^2}{qp\mu_p}$
Joule heat in conductive insulators	JouleHeat	$\frac{ \vec{J}_{CI} ^2}{\sigma}$
Recombination heat	RecombinationHeat	$qR_{\text{total}}((\Phi_p + TP_p) - (\Phi_n + TP_n))$
Net recombination heat	netRecombinationHeat	$qR_{\text{net},p}(\Phi_p + TP_p) - qR_{\text{net},n}(\Phi_n + TP_n)$
Thomson plus Peltier heat [1]	ThomsonHeat	$-\vec{J}_n \cdot T\nabla P_n - \vec{J}_p \cdot T\nabla P_p$
Peltier heat	PeltierHeat	$-T \frac{\partial P_n}{\partial n} \vec{J}_n \cdot \nabla n + \frac{\partial P_p}{\partial p} \vec{J}_p \cdot \nabla p$

Here  $R_{\text{total}} = 0.5(R_{\text{net},p} + R_{\text{net},n})$ .

To plot the lattice heat flux vector  $\kappa\nabla T$ , specify the keyword `lHeatFlux` in the Plot section (see [Device Plots on page 177](#)).

Sentaurus Device reports the maximum, minimum, and average device temperatures in both the plot file and log file whenever the self-heating equation is solved:

- Maximum temperature ( $T_{\max}$ ) in the device as a function of the bias condition
- Minimum temperature ( $T_{\min}$ ) in the device as a function of the bias condition
- Average temperature ( $T_{\text{avg}}$ ) in the device as a function of the bias condition

Device temperatures are not monitored and remain the same as the default lattice temperature, that is, room temperature, if the lattice temperature solution is not requested.

You can also use the current plot feature of Sentaurus Device to extract temperature information in different materials and regions. For example:

```
CurrentPlot {
    Temperature(
        Minimum(Material="Silicon")
```

## Chapter 9: Lattice and Carrier Temperature Equations

### Hydrodynamic Model for Temperatures

```

Maximum(Material="Silicon")
Average(Material="Silicon")
Minimum(Material="Oxide")
Maximum(Material="Oxide")
Average(Material="Oxide")
)
}

```

## Hydrodynamic Model for Temperatures

Deep-submicron devices cannot be described properly using the conventional drift-diffusion transport model. In particular, the drift-diffusion approach cannot reproduce velocity overshoot and often overestimates the impact ionization generation rates. In this case, the hydrodynamic (or energy balance) model provides a good compromise between physical accuracy and computation time.

Since the work of Stratton [2] and Bløtekjær [3], there have been many variations of this model. The full formulation includes convective terms [4]. Sentaurus Device implements a simpler formulation without convective terms. In addition to the Poisson equation (Equation 37) and continuity equations (Equation 57), up to three additional equations for the lattice temperature  $T$  and the carrier temperatures  $T_n$  and  $T_p$  can be solved.

The energy balance equations read:

$$\frac{\partial W_n}{\partial t} + \nabla \cdot \vec{S}_n = \vec{J}_n \cdot \nabla E_C/q + \left. \frac{dW_n}{dt} \right|_{\text{coll}} \quad (76)$$

$$\frac{\partial W_p}{\partial t} + \nabla \cdot \vec{S}_p = \vec{J}_p \cdot \nabla E_V/q + \left. \frac{dW_p}{dt} \right|_{\text{coll}} \quad (77)$$

$$\frac{\partial W_L}{\partial t} + \nabla \cdot \vec{S}_L = \left. \frac{dW_L}{dt} \right|_{\text{coll}} \quad (78)$$

where the energy fluxes are:

$$\vec{S}_n = -\frac{5r_n\lambda_n}{2} \frac{kT_n}{q} \vec{J}_n + f_n^{\text{hf}} \hat{\kappa}_n \nabla T_n \quad (79)$$

$$\vec{S}_p = -\frac{5r_p\lambda_p}{2} \frac{-kT_p}{q} \vec{J}_p + f_p^{\text{hf}} \hat{\kappa}_p \nabla T_p \quad (80)$$

$$\vec{S}_L = -\hat{\kappa}_L \nabla T_L \quad (81)$$

$$\hat{\kappa}_n = \frac{k^2}{q} n \mu_n T_n \quad (82)$$

$$\hat{\kappa}_p = \frac{k^2}{q} p \mu_p T_p \quad (83)$$

## Chapter 9: Lattice and Carrier Temperature Equations

### Hydrodynamic Model for Temperatures

and  $\lambda_n$  and  $\lambda_p$  are defined as for [Equation 64](#) and [Equation 65](#). The parameters  $r_n$ ,  $r_p$ ,  $f_n^{\text{hf}}$ , and  $f_p^{\text{hf}}$  are accessible in the parameter file of Sentaurus Device. Different values of these parameters can significantly influence the physical results, such as velocity distribution and possible spurious velocity peaks. By changing these parameters, Sentaurus Device can be tuned to a very wide set of hydrodynamic or energy balance models as described in the literature. The default parameter values of Sentaurus Device are:

$$r_n = r_p = 0.6 \quad (84)$$

$$f_n^{\text{hf}} = f_p^{\text{hf}} = 1 \quad (85)$$

By changing the constants  $f_n^{\text{hf}}$  and  $r_n$ , you can change the convective contribution and the diffusive contributions independently. With the default set of transport parameters of Sentaurus Device, the prefactor of the diffusive term has the form:

$$\kappa_n = \frac{3}{2} \cdot \frac{k^2 \lambda_n}{q} n \mu_n T_n \quad (86)$$

The collision terms are expressed by this set of equations:

$$\left. \frac{\partial W_n}{\partial t} \right|_{\text{coll}} = -H_n - \xi_n \frac{W_n - W_{n0}}{\tau_{en}} \quad (87)$$

$$\left. \frac{\partial W_p}{\partial t} \right|_{\text{coll}} = -H_p - \xi_p \frac{W_p - W_{p0}}{\tau_{ep}} \quad (88)$$

$$\left. \frac{\partial W_L}{\partial t} \right|_{\text{coll}} = H_L + \xi_n \frac{W_n - W_{n0}}{\tau_{en}} + \xi_p \frac{W_p - W_{p0}}{\tau_{ep}} \quad (89)$$

Here,  $H_n$ ,  $H_p$ , and  $H_L$  are the energy gain/loss terms due to generation–recombination processes. The expressions used for these terms are based on approximations [5] and have the following form for the major generation–recombination phenomena:

$$H_n = 1.5kT_n(R_{\text{net}}^{\text{SRH}} + R_{\text{net}}^{\text{rad}} + R_{n,\text{net}}^{\text{trap}}) - E_{g,\text{eff}}(R_n^A - G_n^{\text{ii}}) - \alpha(\hbar\omega - E_{g,\text{eff}})G^{\text{opt}} \quad (90)$$

$$H_p = 1.5kT_p(R_{\text{net}}^{\text{SRH}} + R_{\text{net}}^{\text{rad}} + R_{p,\text{net}}^{\text{trap}}) - E_{g,\text{eff}}(R_p^A - G_p^{\text{ii}}) - (1 - \alpha)(\hbar\omega - E_{g,\text{eff}})G^{\text{opt}} \quad (91)$$

$$H_L = [R_{\text{net}}^{\text{SRH}} + 0.5(R_{n,\text{net}}^{\text{trap}} + R_{p,\text{net}}^{\text{trap}})](E_{g,\text{eff}} + 1.5kT_n + 1.5kT_p) \quad (92)$$

where:

- $R_{\text{net}}^{\text{SRH}}$  is the Shockley–Read–Hall (SRH) recombination rate (see [Shockley–Read–Hall Recombination on page 473](#)).
- $R_{\text{net}}^{\text{rad}}$  is the radiative recombination rate (see [Radiative Recombination Model on page 488](#)).
- $R_n^A$  and  $R_p^A$  are Auger recombination rates related to electrons and holes (see [Auger Recombination Model on page 489](#)).
- $\hbar\omega$  is the photon energy (see [Hydrodynamic Model Parameters on page 254](#)).

## Chapter 9: Lattice and Carrier Temperature Equations

### Hydrodynamic Model for Temperatures

- $\alpha$  is a dimensionless parameter that describes how the surplus energy of the photon splits between the bands (see [Hydrodynamic Model Parameters on page 254](#)).
- $G_n^{ii}$  and  $G_p^{ii}$  are impact ionization rates (see [Avalanche Generation on page 494](#)).
- $R_{n,\text{net}}^{\text{trap}}$  and  $R_{p,\text{net}}^{\text{trap}}$  are the recombination rates through trap levels (see [Chapter 17 on page 543](#)).
- $G^{\text{opt}}$  is the optical generation rate (see [Chapter 21 on page 648](#)).

Surface recombination is taken into account in a way similar to bulk SRH recombination. Usually, the influence of  $H_n$ ,  $H_p$ , and  $H_L$  is small, so they are not activated by default. To take these energy sources into account, the keyword `RecGenHeat` must be specified in the `Physics` section.

The energy densities  $W_n$ ,  $W_p$ , and  $W_L$  are given by:

$$W_n = nw_n = n \frac{3kT_n}{2} \quad (93)$$

$$W_p = pw_p = p \frac{3kT_p}{2} \quad (94)$$

$$W_L = c_L T \quad (95)$$

The corresponding equilibrium energy densities are:

$$W_{n0} = nw_0 = n \frac{3kT}{2} \quad (96)$$

$$W_{p0} = pw_0 = p \frac{3kT}{2} \quad (97)$$

The parameters  $\xi_n$  and  $\xi_p$  in [Equation 87](#) to [Equation 89](#) improve numeric stability. They speed up relaxation for small densities and they approach 1 for large densities:

$$\xi_n = 1 + \frac{n_{\min}}{n} \frac{n_0}{n_{\min}} \max[0, (T - T_n)/100\text{ K}] \quad (98)$$

and likewise for  $\xi_p$ . Here,  $n_{\min}$  and  $n_0$  are adjustable small density parameters.

## Hydrodynamic Model Parameters

The default set of transport coefficients ([Equation 84](#) and [Equation 85](#)) can be changed in the parameter file. The coefficients  $r$  and  $f$  are specified in the `EnergyFlux` and `HeatFlux` parameter sets, respectively. Energy relaxation times  $\tau_{en}$  and  $\tau_{ep}$  can be modified in the `EnergyRelaxationTime` parameter set.

$\alpha$  in [Equation 90](#) and [Equation 91](#) divides the contribution of the optical generation rate into the energy gain or loss terms  $H_n$  and  $H_p$ . `OptGenOffset` specifies  $\alpha$ , which can take values between 0 and 1 (default is 0.5). The angular frequency  $\omega$  in [Equation 90](#) and

## Chapter 9: Lattice and Carrier Temperature Equations

### Hydrodynamic Model for Temperatures

[Equation 91](#) corresponds to the wavelength specified to compute the optical generation rate. This wavelength is defined by `OptGenWavelength` if the optical generation is loaded from a file (see [Optical AC Analysis on page 770](#)).

Both `OptGenOffset` and `OptGenWavelength` are options to `RecGenHeat` (see [Table 236 on page 1617](#)).

The parameters  $n_{\min}$  and  $n_0$  in [Equation 98](#) are set with `RelTermMinDensity` and `RelTermMinZeroDensity`, respectively, in the global `Math` section. The default values for  $n_{\min}$  and  $n_0$  are  $10^3 \text{ cm}^{-3}$  and  $2 \times 10^8 \text{ cm}^{-3}$ , respectively.

---

## Using the Hydrodynamic Model

To activate the hydrodynamic model, you must specify the keyword `Hydrodynamic` in the `Physics` section. If only one carrier temperature equation is to be solved, `Hydrodynamic` must be specified with the appropriate parameter, either `Hydrodynamic(eTemperature)` or `Hydrodynamic(hTemperature)`. If the hydrodynamic model is not activated for a particular carrier type, Sentaurus Device merges the temperature for this carrier with the lattice temperature. That is, these temperatures are equal, and their heating terms (see the right-hand sides of [Equation 76](#), [Equation 77](#), and [Equation 78](#)) are added.

In addition, you must specify the `eTemperature`, `hTemperature`, and `Temperature` keywords inside the `Coupled` statement of the `Solve` section (see [Coupled Statement on page 191](#)) to actually solve a carrier temperature equation or the lattice temperature equation. Temperatures remain fixed during `Solve` statements in which their equation is not solved.

By default, the energy conservation equations of Sentaurus Device do not include generation–recombination heat sources. To activate them, the keyword `RecGenHeat` must be specified in the `Physics` section.

To plot the electron, hole, and lattice heat flux vectors  $\vec{S}_n$ ,  $\vec{S}_p$ ,  $\vec{S}_L$  (see [Equation 79](#), [Equation 80](#), and [Equation 81](#)), specify the corresponding keywords `eHeatFlux`, `hHeatFlux`, and `lHeatFlux` in the `Plot` section (see [Device Plots on page 177](#)).

To plot  $J_n \cdot \nabla E_C/q$  and  $J_p \cdot \nabla E_V/q$  (see [Equation 76](#) and [Equation 77](#)), specify the corresponding keywords `eJEheat` and `hJEHeat`.

To plot  $dW_n/dt|_{\text{coll}}$ ,  $dW_p/dt|_{\text{coll}}$ , and  $dW_L/dt|_{\text{coll}}$  (see [Equation 76](#), [Equation 77](#), and [Equation 78](#)), specify the corresponding keywords `eColHeat`, `hColHeat`, and `lColHeat`.

To plot  $H_n$ ,  $H_p$ , and  $H_L$  (see [Equation 90](#), [Equation 91](#), and [Equation 92](#)), specify the corresponding keywords `eRecGenHeat`, `hRecGenHeat`, and `lRecGenHeat`.

## Chapter 9: Lattice and Carrier Temperature Equations

### Numeric Parameters for Temperature Equations

---

## Numeric Parameters for Temperature Equations

This section presents the numeric parameters for temperature equations.

---

### Validity Ranges for Lattice and Carrier Temperatures

Lower and upper limits for the lattice temperature and the carrier temperatures exist.

The allowed temperature ranges are specified (in K) by `lT_Range=(<float> <float>)` (defaults 50 K and 5000 K) and `cT_Range=(<float> <float>)` (defaults 10 K and 80000 K). These ranges apply both to the temperatures during the Newton iterations and to the final results.

---

### Scaling of Lattice Heat Generation

For the lattice heat equation, you have the possibility to disable the heat term by using:

```
Physics ( Region="Bulk" ) { ... HeatPreFactor = 0. }
```

This reduces or even eliminates the strong coupling, given by the Joule heat, of the lattice heat equation with the carrier continuity equations, and leads in general to smooth temperature profiles. This simplified coupled system of carrier transport and lattice temperature might converge easier than the fully coupled system including the heat term. Instead of ramping the fully coupled system to a required bias condition, it might be easier to ramp first the simplified coupled system to the required bias condition, and ramp subsequently the `HeatPreFactor` from zero to one.

#### Note:

Due to the strong nonlinearity of semiconductor equations, which allows multiple solutions in general, and especially for examples with strong generation–recombination rates, the `HeatPreFactor` ramping might lead to a different but still valid solution.

In the command file, this second ramp could be:

```
Quasistationary (
    Goal { Region="Bulk" ModelParameter="Physics/HeatPreFactor" Value=1 }
) { Coupled { Poisson Electron Hole Temperature } }
```

#### Note:

The `Goal` statement requires the specification of a `Region`, and a `Physics` section for the corresponding region must be specified in the command file.

## References

- [1] K. Kells, *General Electrothermal Semiconductor Device Simulation*, Series in Microelectronics, vol. 37, Konstanz, Germany: Hartung-Gorre, 1994.
- [2] R. Stratton, "Diffusion of Hot and Cold Electrons in Semiconductor Barriers," *Physical Review*, vol. 126, no. 6, pp. 2002–2014, 1962.
- [3] K. Bløtekjær, "Transport Equations for Electrons in Two-Valley Semiconductors," *IEEE Transactions on Electron Devices*, vol. ED-17, no. 1, pp. 38–47, 1970.
- [4] A. Benvenuti *et al.*, "Evaluation of the Influence of Convective Energy in HBTs Using a Fully Hydrodynamic Model," in *IEDM Technical Digest*, Washington, DC, USA, pp. 499–502, December 1991.
- [5] D. Chen *et al.*, "Dual Energy Transport Model with Coupled Lattice and Carrier Temperatures," in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 5, Vienna, Austria, pp. 157–160, September 1993.

# 10

## Boundary Conditions

---

*This chapter describes the boundary conditions available in Sentaurus Device.*

This chapter describe the properties of electrical contacts, thermal contacts, floating contacts, as well as boundary conditions at other borders of a domain where an equation is solved. Continuity conditions (how the solutions of one equation on two neighboring regions are matched at the region interface) are discussed elsewhere (see, for example, [Dipole Layer on page 230](#)). This chapter is restricted to the boundary conditions for the equations presented in [Chapter 7 on page 229](#), [Chapter 8 on page 238](#), and [Chapter 9 on page 245](#). Boundary conditions for less important equations are discussed with the equations themselves (see, for example, [Chapter 11 on page 291](#)).

---

### Electrical Boundary Conditions

This section discusses electrical boundary conditions.

---

#### Ohmic Contacts

By default, contacts on semiconductors are Ohmic, with no resistance. When connected to a circuit node, the resistance is activated by:

Math { MixedModeContactResistance=<float> }

The default is 0. In this case, a 0.001 resistance is added to the circuit node, which is not scaled with `AreaFactor`. If `<float> > 0`, then `<float> * AreaFactor` is added to the circuit node. The `AreaFactor` comes solely from the `Physics` section. The one from the `Electrode` section is ignored.

Charge neutrality and equilibrium are assumed at Ohmic contacts:

$$n_0 - p_0 = N_D - N_A \quad (99)$$

$$n_0 p_0 = n_{i,eff}^2 \quad (100)$$

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

For Boltzmann statistics, these conditions can be expressed analytically:

$$\phi = \phi_F + \frac{kT}{q} \operatorname{asinh} \frac{N_D - N_A}{2n_{i,\text{eff}}} \quad (101)$$

$$n_0 = \sqrt{\frac{(N_D - N_A)^2}{4} + n_{i,\text{eff}}^2} + \frac{N_D - N_A}{2} \quad p_0 = \sqrt{\frac{(N_D - N_A)^2}{4} + n_{i,\text{eff}}^2} - \frac{N_D - N_A}{2} \quad (102)$$

where  $n_0, p_0$  are the electron and hole equilibrium concentrations, and  $\phi_F$  is the Fermi potential at the contact (which equals the applied voltage if it is not a resistive contact; see [Resistive Contacts on page 270](#)). For Fermi statistics, Sentaurus Device computes the equilibrium solution numerically.

By default,  $n = n_0, p = p_0$  are applied for concentrations at the Ohmic contacts. If the electron or hole recombination velocity is specified, Sentaurus Device uses the following current boundary conditions:

$$\vec{J}_n \cdot \hat{n} = qv_n(n - n_0) \quad \vec{J}_p \cdot \hat{n} = -qv_p(p - p_0) \quad (103)$$

where  $v_n, v_p$  are the electron and hole recombination velocities. In the command file, the recombination velocities can be specified as:

```
Electrode { ...
  { Name="Emitter" Voltage=0 eRecVelocity = v_n hRecVelocity = v_p }
}
```

#### Note:

If the values of the electron and hole recombination velocities equal zero ( $v_n = 0, v_p = 0$ ), then  $\vec{J}_n \cdot \hat{n} = 0, \vec{J}_p \cdot \hat{n} = 0$ . That is, the electrode only defines the electrostatic potential (which is useful for SOI devices).

By specifying `Poisson=Neumann` for an electrode, the boundary condition for the Poisson equation at a contact can be switched to a homogeneous Neumann boundary condition, that is, [Equation 101](#) is not applied.

---

## Modified Ohmic Contacts

Ohmic contacts, as described in [Ohmic Contacts on page 258](#), can often lead to incorrect results around p-n junctions and heterointerfaces. The main cause is the charge neutrality condition imposed at the contact vertices located within the charged depletion region of the p-n junction or heterointerface. Ideally, the carrier densities at such contacts should be an extension of bulk densities with no sharp jump along the contact.

Sentaurus Device supports an alternative for Ohmic contacts that does not impose the charge neutrality condition. In this approach, the equilibrium nonlinear Poisson equation is solved at the beginning of the simulation with the Ohmic contacts under consideration

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

removed (see [Equilibrium Solution on page 231](#) for setting numeric parameters to obtain the equilibrium solution). The equilibrium electrostatic potential  $\phi_{eq}$  obtained in this way is used instead of the built-in potential  $\phi_{bi} = kT \operatorname{asinh}[(N_D - N_A)/2n_{i,eff}]/q$  on the vertices of the Ohmic contacts under consideration. The boundary conditions for the continuity equations remain formally the same as in the case of Ohmic contacts but with the built-in potential  $\phi_{bi}$  replaced by  $\phi_{eq}$ .

You can set the modified Ohmic contacts in the `Electrode` section with the keyword `EqOhmic`:

```
Electrode {  
    ...  
    {Name="drain" Voltage=0 EqOhmic}  
}
```

**Note:**

The feature is supported only for isothermal simulations.

---

## Contacts on Insulators

For contacts on insulators (for example, gate contacts), the electrostatic potential is:

$$\phi = \phi_F - \Phi_{MS} \quad (104)$$

where  $\phi_F$  is the Fermi potential at the contact (which equals the applied voltage if it is not a resistive contact; see [Resistive Contacts on page 270](#)), and  $\Phi_{MS}$  is the workfunction difference between the metal and an intrinsic reference semiconductor.

**Note:**

If an Ohmic contact touches both an insulator and a semiconductor, the electrostatic potential along the contact can be discontinuous because the boundary conditions ([Equation 101](#) and [Equation 104](#)) usually contradict.

[Equation 101](#) takes precedence over [Equation 104](#).

$\Phi_{MS}$  can be specified in the `Electrode` section, either directly with `Barrier`, by specifying the metal workfunction with `WorkFunction`, or by specifying the electrode material:

```
Electrode { ...  
    { Name="Gate" Voltage=0 Barrier = 0.55 }  
}  
Electrode { ...  
    { Name="Gate" Voltage=0 WorkFunction = 5 }  
}  
Electrode { ...  
    { Name="Gate" Voltage=0 Material="Gold" }  
}
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

In the last case, the value for the workfunction is obtained from the parameter file:

```
Material = "Gold" {  
    Bandgap { WorkFunction = 5 }  
}
```

To emulate a poly gate, a semiconductor material and doping concentration can be specified in the `Electrode` section:

```
Electrode { ...  
    { Name="Gate" Voltage=0 Material="Silicon" (N = 5e19) }  
}
```

where `N` is used to specify the doping in an n-type semiconductor material. The built-in potential is approximated by  $(kT/q)\ln(N/n_i)$ . A p-type doping can be selected, similarly, by the letter `P`. If the value of the doping concentration is not specified (only `N` or `P`), the Fermi potential of the electrode is assumed to equal the conduction or valence band edge.

---

## Schottky Contacts

The physics of Schottky contacts is considered in detail in [1] and [2]. In this section, a typical model for Schottky contacts [3] is considered. The following boundary conditions hold:

$$\phi = \phi_F - \Phi_B + \frac{kT}{q} \ln \frac{N_C}{n_{i,eff}} \quad (105)$$

$$\vec{J}_n \cdot \hat{n} = qv_n(n - n_0^B) \quad \vec{J}_p \cdot \hat{n} = -qv_p(p - p_0^B) \quad (106)$$

$$n_0^B = N_C \exp \left( \frac{-q\Phi_B}{kT} \right) \quad p_0^B = N_V \exp \left( \frac{-E_{g,eff} + q\Phi_B}{kT} \right) \quad (107)$$

where:

- $\phi_F$  is the Fermi potential at the contact that is equal to an applied voltage  $V_{\text{applied}}$  if it is not a resistive contact (see [Resistive Contacts on page 270](#)).
- $\Phi_B$  is the barrier height (the difference between the contact workfunction and the electron affinity of the semiconductor in n-type semiconductors, or the difference between the band gap and the barrier as defined for n-type semiconductors in the case of p-type semiconductors).
- $v_n$  and  $v_p$  are the thermionic emission velocities.
- $n_0^B$  and  $p_0^B$  are the equilibrium densities.

The default values for the thermionic emission velocities  $v_n$  and  $v_p$  are  $2.573 \times 10^6$  cm/s and  $1.93 \times 10^6$  cm/s, respectively.

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

The recombination velocities can be set in the `Electrode` section. For example:

```
Electrode { ...
  { Name="Gate" Voltage=0 Schottky Barrier =  $\Phi_B$  eRecVelocity =  $v_n$ 
    hRecVelocity =  $v_p$  }
}
```

#### Note:

The `Barrier` specification might produce incorrect results if the Schottky contact is connected to several different semiconductors. In such cases, use `WorkFunction` instead of `Barrier`. See [Contacts on Insulators on page 260](#).

Sentaurus Device also allows thermionic emission velocities  $v_n$  and  $v_p$  to be defined as functions of lattice temperature:

$$v_n(T) = v_n(300 \text{ K}) \sqrt{\frac{m_n(300 \text{ K})}{m_n(T)}} \sqrt{\frac{T}{300 \text{ K}}} \quad v_p(T) = v_p(300 \text{ K}) \sqrt{\frac{m_p(300 \text{ K})}{m_p(T)}} \sqrt{\frac{T}{300 \text{ K}}} \quad (108)$$

where:

- $m_n$  and  $m_p$  are the temperature-dependent electron and hole density-of-states (DOS) effective masses.
- $v_n(300 \text{ K})$  and  $v_p(300 \text{ K})$  are the recombination velocities at 300 K specified in the `Electrode` section by `eRecVelocity` and `hRecVelocity`.

The default values for the thermionic emission velocities  $v_n(300 \text{ K})$  and  $v_p(300 \text{ K})$  are  $2.573 \times 10^6 \text{ cm/s}$  and  $1.93 \times 10^6 \text{ cm/s}$ , respectively.

To activate the temperature-dependent recombination velocity, specify `eRecVel(TempDep)`, or `hRecVel(TempDep)`, or `RecVel(TempDep)` in the electrode-specific `Physics` section:

```
Physics (Electrode = "cathode") { RecVel(TempDep) }
```

## Fermi-Level Pinning at Schottky Contacts

At a metal semiconductor contact or interface, there is a charge transfer between the metal and the interface states located inside the semiconductor bandgap. The charge transfer creates an interfacial dipole, which can change the Schottky barrier. This phenomenon, known as Fermi-level pinning, can be modeled by a Schottky pinning parameter  $S$ .

The Schottky potential barrier accounting for the correction due to the charge transfer between the metal and the interface states in the semiconductor bandgap (pinning effect) is modeled as:

$$\Phi_B = S \Phi_M - \frac{E_{CNL}}{q} + \frac{E_{CNL}}{q} - \frac{\chi}{q} \quad (109)$$

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

where:

- $\Phi_M$  is the metal workfunction in V.
- $E_{CNL}$  is the charge neutrality level with respect to the vacuum level in eV.
- $S$  is the Schottky pinning parameter. For  $S = 1$  (no pinning), there is no charge transfer, and the Schottky barrier becomes the classical Schottky limit. For  $S = 0$  (strong pinning), you obtain the Bardeen limit where the metal Fermi level is pinned by the interface states at  $E_{CNL}$ .

The models available in Sentaurus Device for the Schottky pinning parameter  $S$  are the Sze model and the Mönch model.

The Sze model is defined as [4]:

$$S = \frac{1}{1 + \frac{q^2(N_I d)}{\epsilon \epsilon_0}} \quad (110)$$

where  $\epsilon$  is the semiconductor permittivity,  $N_I$  is the density of interface states per unit energy, and  $d$  is their extent into the semiconductor. The parameters  $N_I$  and  $d$  are mole fraction dependent and can be modified in the Schottky section of the parameter file.

The Mönch model, which is the default, is an empirical formula and is defined as [5]:

$$S = \frac{1}{1 + A(\epsilon_\infty - 1)^B} \quad (111)$$

where  $\epsilon_\infty$  is the high-frequency limit of the permittivity (set to 12 for most materials), and  $A$  and  $B$  are mole fraction-adjustable parameters that you can modify in the Schottky section of the parameter file. You can set  $\epsilon_\infty$  in the Epsilon\_Inf section of the parameter file:

```
Epsilon_Inf { epsilon_inf = 12 }
```

The charge neutrality level  $E_{CNL}$  is also a mole fraction-dependent parameter adjustable in the parameter file.

#### Note:

In general,  $\epsilon_\infty$  is tabulated for standard materials, so  $S$  can be reliably determined from Equation 111 for those materials. Using the  $S$  equivalency between Equation 110 and Equation 111 values for  $N_I d$  can be inferred.

To activate the pinning effect at the Schottky contacts, specify the Pinning keyword in the Electrode section, with the model defined as an option for the keyword Model:

```
Electrode { ...
  { name = "cnt1" voltage = 0 Schottky(Pinning(Model="Sze"))
    Workfunction=4.75 }
}
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

If the keyword `Model` is omitted, the Mönch model is used. The pinning model specification in the `Electrode` section has a higher priority. If the pinning model is specified in both the `Electrode` section and the `electrode Physics` section, the specification in the `Electrode` section is chosen.

The pinning model parameters in [Equation 110](#) and [Equation 111](#) are accessible in the `Schottky` subsection in the `Electrode` section of the parameter file.

Table 29 Parameters for Schottky pinning

Symbol	Parameter name	Default value	Unit
A	Pinning_A	0.1	1
B	Pinning_B	2	1
$N_I$	Pinning_Nint	$10^{10}$	$\text{cm}^{-2} \text{ eV}^{-1}$
d	Pinning_d	$2 \times 10^{-7}$	cm
$E_{\text{CNL}}$	Pinning_CNL	5.01964	eV

For example, to specify a mole fraction-dependent `CNL` on the contact `drain`, which is on the top of a mole fraction-dependent SiGe region `r3` defined as:

```
Physics(Region = "r3") {    # Ge(x)Si(1-x)
* Mole-dependent material: SiGe(x=0) = Silicon
* Mole-dependent material: SiGe(x=1) = Germanium
    MoleFraction(
        xFraction=0.35
    )
}
```

specify in the parameter file something like:

```
Region = "r3" {
    Schottky {
        Xmax(0)=0.0
        Xmax(1)=0.8
        Xmax(2)=1.0

        Electrode = "drain" {
            Pinning_CNL(0) = 3.5
            Pinning_CNL(1) = 3.7
            Pinning_CNL(2) = 3.8
        }
    }
}
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

See [Resistive Contacts on page 270](#) for a detailed description on how to specify mole-fraction parameters on contacts and interfaces.

## Barrier Lowering at Schottky Contacts

The barrier-lowering model for Schottky contacts can account for different physical mechanisms. The most important one is the image force [3], but it can also model tunneling and dipole effects.

The barrier lowering seen by the electron being emitted from metal into the conduction band and for the hole being emitted into the valence are given by:

$$\Delta\Phi_{B,e}(F) = \begin{cases} \frac{\vec{F} \cdot \hat{n}}{F_0} \left[ \frac{pp_{1,e}}{F_0} + \frac{aa_{2,e}}{F_0} \cdot \frac{\vec{F} \cdot \hat{n}}{F_0} \right] pp_{2,e} & \text{if } \vec{F} \cdot \hat{n} > 0 \\ 0 & \text{if } \vec{F} \cdot \hat{n} \leq 0 \end{cases} \quad (112)$$

$$\Delta\Phi_{B,h}(F) = \begin{cases} \frac{\vec{F} \cdot \hat{n}}{F_0} \left[ \frac{pp_{1,h}}{F_0} + \frac{aa_{2,h}}{F_0} \cdot \frac{\vec{F} \cdot \hat{n}}{F_0} \right] pp_{2,h} & \text{if } \vec{F} \cdot \hat{n} \leq 0 \\ 0 & \text{if } \vec{F} \cdot \hat{n} > 0 \end{cases} \quad (113)$$

where:

- $\vec{F} \cdot \hat{n}$  is the normal component of the electric field on the local exterior normal pointing from semiconductor region to metal contact,  $F_0 = 1 \text{ Vcm}^{-1}$ .
- $aa_{1,e}, aa_{2,e}, pp_{1,e}, pp_{2,e}, aa_{1,h}, aa_{2,h}, pp_{1,h}, pp_{2,h}$  are the model coefficients that can be specified in the parameter file of Sentaurus Device. Their default values are  $aa_{1,e} = aa_{1,h} = 2.6 \times 10^{-4} \text{ eV}$ ,  $pp_{1,e} = pp_{1,h} = 0.5$ ,  $aa_{2,e} = aa_{2,h} = 0 \text{ eV}$ , and  $pp_{2,e} = pp_{2,h} = 1$ .

The final value of the Schottky barrier is computed as  $\Phi_B - \Delta\Phi_{B,e}(F)$  for the electrons in the conduction band and as  $\Phi_B + \Delta\Phi_{B,h}(F)$  for the holes in the valence band. The barrier lowering also affects the equilibrium concentration of electrons  $n_0^B$  and holes  $p_0^B$  corresponding to its formula ([Equation 107](#)), through the correction in barrier  $\Phi_B$  described above.

In addition, Sentaurus Device implements a simplified model for barrier lowering. In this case, the barrier lowering is:

$$\Delta\Phi_B(F) = \begin{cases} a_1 \left[ \frac{F}{F_0} \frac{p_1}{F_0} \cdot \frac{F_{eq}}{F_0} p_{1,eq} \right] + a_2 \left[ \frac{F}{F_0} \frac{p_2}{F_0} \cdot \frac{F_{eq}}{F_0} p_{2,eq} \right] & \text{if } F > \eta F_{eq} \\ 0 & \text{if } F \leq \eta F_{eq} \end{cases} \quad (114)$$

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

where:

- $F_0 = 1 \text{ Vcm}^{-1}$ .
- $F_{\text{eq}}$  is the equilibrium electric field (see [Equilibrium Solution on page 231](#) for setting numeric parameters to obtain the equilibrium solution).
- $a_1, a_2, p_1, p_{1,\text{eq}}, p_2$ , and  $p_{2,\text{eq}}$  are the model coefficients that can be specified in the parameter file of Sentaurus Device. Their default values are  $a_1 = 2.6 \times 10^{-4} \text{ eV}$ ,  $a_2 = 0 \text{ eV}$ ,  $p_1 = p_{1,\text{eq}} = 0.5$ , and  $p_2 = p_{2,\text{eq}} = 1$ .

For fields smaller than  $\eta F_{\text{eq}}$  (where  $\eta$  is one by default), barrier lowering is zero. The final value of the Schottky barrier is computed as  $\Phi_B - \Delta\Phi_B(F)$  for n-doped contacts and  $\Phi_B + \Delta\Phi_B(F)$  for p-doped contacts, because a difference between the metal workfunction and the valence band must be considered if holes are majority carriers. This model does not account for the direction of the electric field that determines in which band the barrier forms. For example, assume a Schottky contact on an n-type semiconductor with Schottky barrier 0.8 V. The flat band condition is at a forward bias of approximately 0.8 V. For a reverse bias and forward bias less than 0.8 V, the barrier is in the conduction band, so the barrier lowering is applied correctly for electrons. On the other hand, for a forward bias greater than 0.8 V, the barrier is now in the valence band, so the barrier lowering should be applied to holes, not to electrons as the model does. In this case, this simplified model fails. This model is tuned to work properly for reverse biases, where the barrier lowering produces a sizable change in I-V characteristics of the Schottky device.

You can use the `weight` parameter to combine one of the barrier models with the barrier lowering with interfacial insulator layer model described by [Equation 115](#). If the barrier-lowering model is the only model specified (no interfacial insulator layer model), the value of `weight` is 1 regardless of the value specified in the parameter file by `c_weight`.

*Table 30 Parameters for barrier-lowering models*

Symbol	Parameter name	Default value	Unit
$aa_{1,e}$	<code>aa1</code>	$a_1 = 2.6 \times 10^{-4}$	eV
$aa_{1,h}$	<code>aa1</code>	$a_1 = 2.6 \times 10^{-4}$	eV
$aa_{2,e}$	<code>aa2</code>	0	eV
$aa_{2,h}$	<code>aa2</code>	0	eV
$pp_{1,e}$	<code>pp1</code>	0.5	1
$pp_{1,h}$	<code>pp1</code>	0.5	1

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

*Table 30 Parameters for barrier-lowering models (Continued)*

Symbol	Parameter name	Default value	Unit
$pp_{2,e}$	pp2	1	1
$pp_{2,h}$	pp2	1	1
$a_1$	a1	$2.6 \times 10^{-4}$	eV
$a_2$	a2	0	eV
$p_1$	p1	0.5	1
$p_2$	p2	1	1
$p_{1,\text{eq}}$	p1_eq	0.5	1
$p_{2,\text{eq}}$	p2_eq	1	1
$\eta$	eta	1	1
weight	c_weight	1	1

To activate the barrier-lowering model, specify `BarrierLowering` in the electrode-specific `Physics` section for the simplified model:

```
Physics(Electrode = "Gate") { BarrierLowering }
```

For the complete model, specify:

```
Physics(Electrode = "Gate") { BarrierLowering(Full) }
```

To specify parameters of the model, create the following parameter set:

```
Electrode = "Gate" {
    BarrierLowering {
        a1 = a1                                * [eV]
        p1 = p1                                * [1]
        p1_eq = p1_eq                           * [1]
        a2 = a2                                * [eV]
        p2 = p2                                * [1]
        p2_eq = p2_eq                           * [1]
        eta = η                                 * [1]
        aa1 = aa1_e , aa1_h                   * [eV]
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

```

pp1 = pp1,e, pp1,h * [1]
aa2 = aa2,e, aa2,h * [eV]
pp2 = pp2,e, pp2,h * [1]
}
}

```

To emulate the barrier-lowering effect at a metal-semiconductor contact, with a thin virtual interfacial layer between the metal and the semiconductor, Sentaurus Device supports another barrier-lowering model that takes into account such an insulating interface layer. The model [6] assumes that a finite insulator layer is present between the metal and the semiconductor, and it includes both images with respect to the metal–insulator and insulator–semiconductor interfaces.

Sentaurus Device implements the barrier lowering with interfacial insulating layer model described in detail in [6] as:

$$\Delta\Phi_B(d_0) = \begin{cases} \frac{q}{16\pi\epsilon_{ins}} \frac{1}{d_\epsilon} + \frac{\beta}{t_{ox}-d_\epsilon} & 0 \leq d_0 \leq d_\epsilon \\ \frac{q}{16\pi\epsilon_{ins}} \frac{1}{d_0} + \frac{\beta}{t_{ox}-d_0} & d_\epsilon < d_0 \leq t_{ox}-d_\epsilon \\ \frac{q}{16\pi\epsilon_{ins}} \frac{1}{t_{ox}-d_\epsilon} + \frac{\beta}{d_\epsilon} - A \frac{t_{ox}-d_\epsilon}{2d_\epsilon} + \frac{A}{2d_\epsilon} d_0 & t_{ox}-d_\epsilon < d_0 \leq t_{ox}+d_\epsilon \\ \frac{q}{16\pi\epsilon_{sem}} \frac{1}{d_0} + \frac{\beta}{t_{ox}-d_0} & d_0 > t_{ox}+d_\epsilon \end{cases} \quad (115)$$

where:

- $\beta = \frac{\epsilon_{sem} - \epsilon_{ins}}{\epsilon_{sem} + \epsilon_{ins}}$ .
- $A = \frac{q}{16\pi\epsilon_{sem}} \frac{1}{t_{ox}+d_\epsilon} - \frac{\beta}{d_\epsilon} - \frac{q}{16\pi\epsilon_{ins}} \frac{1}{t_{ox}-d_\epsilon} + \frac{\beta}{d_\epsilon}$ .
- $\epsilon_{sem}$  is the semiconductor layer permittivity.
- $\epsilon_{ins}$  is the insulator layer permittivity.
- $t_{ox}$  is the conforming insulator layer thickness.
- $d_o$  is the distance from the metal surface (contact) to the point where the barrier lowering is evaluated.
- $d_\epsilon$  is a computational-adjustable parameter used to handle critical points in the model.

The second branch in Equation 115 represents the barrier lowering due to the image-force potential of the electron in the insulator layer, where a numeric adjustment is applied to the edges of the layer to avoid infinite values. The first branch is introduced to saturate the value of the image-force potential close to the metal surface (contact), which otherwise would

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

become infinite for  $d_o \rightarrow 0$ . The last branch represents the barrier lowering due to the image-force potential of the electron in the semiconductor layer.

To avoid discontinuity and infinite values at the insulator–semiconductor interface, a linear transitional region is introduced (the third branch).

The  $d_\varepsilon$  parameter, which should be smaller than  $t_{\text{ox}}$  for geometric reasons, allows you to adjust the functional shape of the model around the model critical points).

For consistency, when  $t_{\text{ox}} = 0$ , the model reverts to  $\Delta\Phi_B(d_0, d_0 > 0) = q/16\pi\varepsilon_{\text{sem}}d_0$ , which is the typical image-force barrier lowering for a metal–semiconductor interface.

As an option, you can select to compute a barrier-lowering average based on [Equation 115](#) over the distance  $d_o$  from the contact or an average around  $d_o$ .

The barrier lowering with interfacial insulator layer model can be combined as a weighted sum with any of the barrier-lowering models described by [Equation 112](#) (or [Equation 113](#)) or [Equation 114](#), but not with both.

To activate the barrier lowering with interfacial insulator layer model, specify `InsBarrierLowering` in the `Schottky` subsection of the `Electrode` or the electrode-specific `Physics` section:

```
Electrode { ...
  { name = "cnt1" voltage = 0 Schottky(InsBarrierLowering)
    Workfunction=4.75 }
}
Physics(Electrode = "cnt1") { ...
  Schottky(InsBarrierLowering)
}
```

To compute an average based on [Equation 115](#) over the distance  $d_o$ , specify also the keyword `Average`:

```
Physics(Electrode = "Gate") { InsBarrierLowering(Average) }
```

To compute an average based on [Equation 115](#) over a symmetric interval `delta`, centered around  $d_o$ , specify the keyword `DeltaAverage` in the `InsBarrierLowering` section of the command file:

```
Physics(Electrode = "Gate") { InsBarrierLowering(DeltaAverage) }
```

You can adjust the parameter `InsBL_delta` in the `Schottky` section of the parameter file:

```
Electrode = "anode" {
  Schottky {
    InsBL_delta = 1.0000e-08      * [cm]
  }
}
```

The model parameters  $t_{\text{ox}}$ ,  $\varepsilon_{\text{ins}}$ ,  $d_\varepsilon$ ,  $d_0$ , and `nn` can be specified in the `Schottky` section of the parameter file (see [Table 31 on page 270](#)). The parameter `nn` controls the number of

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

equidistant intervals over which the integration is performed using the trapezoidal rule in the case of averaging. The  $t_{ox}$ ,  $\epsilon_{ins}$ , and  $d_\epsilon$  parameters are mole fraction dependent.

You can use the `weight` parameter to combine the barrier lowering with the interfacial insulator layer model with either the barrier model described by [Equation 112](#) (or [Equation 113](#)) or [Equation 114](#). If only the barrier lowering with interfacial insulator layer model is specified (by `InsBarrierLowering` in the electrode-specific `Physics` section), the value of `weight` is 1 regardless of the value specified in the parameter file by `InsBL_c_weight`. If the model is used together with other barrier-lowering models, the `weight` parameters in both models can be adjusted individually such that their sum is 1. The overall barrier-lowering effect would be a weighted sum of the models.

*Table 31 Parameters of barrier lowering with interfacial insulator layer model*

Symbol	Parameter name	Default value	Unit
$t_{ox}$	<code>InsBL_tox</code>	$2 \times 10^{-7}$	cm
$\epsilon_{ins}$	<code>InsBL_epsins</code>	3.9	1
$d_0$	<code>InsBL_d0</code>	$1 \times 10^{-7}$	cm
$d_\epsilon$	<code>InsBL_d_eps</code>	$1 \times 10^{-8}$	cm
$nn$	<code>InsBL_nn</code>	20	1
<code>weight</code>	<code>InsBL_c_weight</code>	1	1
<code>delta</code>	<code>InsBL_delta</code>	$1 \times 10^{-8}$	cm

## Resistive Contacts

By default, contacts have a resistance of 1 mΩ when they are connected to a circuit node, and no resistance otherwise. The resistance can be changed with `Resist` or `DistResist` or both in the `Electrode` section.

The following example defines an emitter as a contact with resistance 1 Ω (assuming that `AreaFactor` is one):

```
Electrode { ...
  { name = "emitter" voltage = 2 Resist=1 }
}
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

The following example defines an emitter as a contact with a distributed resistance of  $0.0002 \Omega\text{cm}^2$  in series with a  $100 \Omega$  lump resistor, with the  $2 \text{ V}$  external voltage applied to the lump resistor:

```
Electrode { ...
  { name = "emitter" voltage = 2 Resist=100 DistResist=2e-4 }
}
```

If  $V_{\text{applied}}$  is applied to the contact through a resistor  $R$  (`Resist` in the `Electrode` section) or distributed resistance  $R_d$  (`DistResist` in the `Electrode` section), there is an additional equation to compute  $\phi_F$  in [Equation 101](#), [Equation 104](#), and [Equation 105](#).

For a distributed resistance,  $\phi_F$  is different for each mesh vertex of the contact and is computed as a solution of the following equation for each contact vertex, self-consistently with the system of all equations:

$$\hat{n} \cdot [\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F) + \vec{J}_D(\phi_F)] = \frac{(V_{\text{applied}} - \phi_F)}{R_d} \quad (116)$$

For a resistor,  $\phi_F$  is a constant over the contact and only one equation per contact is solved self-consistently with the system of all equations:

$$\int_s \hat{n} \cdot [\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F) + \vec{J}_D(\phi_F)] ds = \frac{(V_{\text{applied}} - \phi_F)}{R} \quad (117)$$

where  $s$  is a contact area used in a Sentaurus Device simulation to compute a total current through the contact.

When both a lump resistor  $R$  and a distributed resistance  $R_d$  are specified,  $\phi_F$  is computed for each mesh vertex of the contact as a solution of the following equations, self-consistently with the system of all equations:

$$\hat{n} \cdot [\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F) + \vec{J}_D(\phi_F)] = \frac{(V_{\text{internal}} - \phi_F)}{R_d} \quad (118)$$

$$\frac{V_{\text{applied}} - V_{\text{internal}}}{R} = \int_s \frac{V_{\text{internal}} - \phi_F}{R_d} ds \quad (119)$$

where:

- $V_{\text{internal}}$  depends on  $\phi_F$  along the contact (on all contact vertices).
- $\int_s \frac{V_{\text{internal}} - \phi_F}{R_d} ds$  is the total current on the contact.

For a distributed resistance, a total current through the contact is calculated by summing the contributions for each contact vertex, scaled by a fraction of the contact assigned to the mesh vertex. In the case of contacts overlaying the oxide materials, the vertex fraction of the contact includes the oxide part for vertices at the interface between

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

conducting and insulating materials. This gives rise to a strong sensitivity of terminal currents with respect to mesh spacing. The erroneous contributions from the oxide contact can be excluded by using the `WithDistResistOnSemi` keyword in the global `Math` section.

#### Note:

In 2D simulations,  $R$  must be specified in units of  $\Omega\mu\text{m}$ . In 3D simulations,  $R$  must be specified in units of W.  $R_d$  is given in  $\text{Wcm}^2$  (see [Table 206](#) on [page 1565](#)).

To emulate the behavior of a Schottky contact [7], Schottky contact resistivity at zero bias was derived and a doping-dependent resistivity model of such contacts was obtained. This model is activated for Ohmic contacts by the keyword `DistResist=SchottkyResist` in the `Electrode` section.

The model is expressed as:

$$\begin{aligned} R_d &= R_\infty \frac{300\text{K}}{T_0} \exp\left(\frac{q\Phi_B}{E_0}\right) \\ E_0 &= E_{00} \coth\left(\frac{E_{00}}{kT_0}\right) \\ E_{00} &= \frac{qh}{4\pi} \sqrt{\frac{|N_{D,0} - N_{A,0}|}{\epsilon_s m_t}} \end{aligned} \quad (120)$$

where:

- $\Phi_B$  is the Schottky barrier (in this model, for electrons, this is the difference between the metal workfunction and the electron affinity of the semiconductor; for holes, this is the difference between the valence band energy of the semiconductor and the metal workfunction).
- $R_\infty$  is the Schottky resistance for an infinite doping concentration at the contact (or zero Schottky barrier).
- $\epsilon_s$  is the semiconductor permittivity.
- $m_t$  is the tunneling mass.
- $T_0$  is the device lattice temperature defined in the `Physics` section (see [Table 236](#) on [page 1617](#)).

The parameters  $\Phi_B$ ,  $R_\infty$ , and  $m_t$  can be mole dependent, and they can be specified in the region or material sections of the parameter file. If the parameters are mole independent, they also can be specified directly in the `Electrode` section of the parameter file.

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

For example, if the contact "top" covers two semiconductor regions "reg1" and "reg2", the mole dependency of parameter  $R_\infty$  can be defined as:

```
Region = "reg1" {
    SchottkyResistance {
        Xmax(0) = 0.0
        Xmax(1) = 0.6
        Electrode = "top" {
            Rinf = 2.4000e-09 , 5.2000e-09      # [Ohm*cm^2]
            Rinf(1) = 2.5000e-09 , 5.2000e-09   # [Ohm*cm^2]
        }
    }
}
Region = "reg2" {
    SchottkyResistance {
        Xmax(0) = 0.0
        Xmax(1) = 0.1
        Xmax(2) = 0.6
        Electrode = "top" {
            Rinf = 2.5000e-09 , 5.2000e-09      # [Ohm*cm^2]
            Rinf(1) = 2.6000e-09 , 5.2000e-09   # [Ohm*cm^2]
            Rinf(2) = 2.9000e-09 , 5.2000e-09   # [Ohm*cm^2]
        }
    }
}
```

In the example, the parameter  $R_\infty$  of contact "top" is defined in "reg1" and "reg2" with different mole dependencies.

If the Schottky resistance parameters are not mole fraction dependent, the parameters can be specified regionwise or materialwise in the `SchottkyResistance` section:

```
Region = "reg1" {
    SchottkyResistance {
        Rinf = 2.4000e-09 , 5.2000e-09      # [Ohm*cm^2]
    }
}
```

For backward compatibility, the Schottky resistance parameters that are not mole fraction dependent also can be defined directly in the `Electrode` section of the Schottky resistive contact:

```
Electrode = "gate" {
    # no mole fraction dependency allowed
    SchottkyResistance {
        Rinf = 2.4000e-09 , 5.2000e-09      # [Ohm*cm^2]
        PhiB = 0.6             , 0.51       # [eV]
    }
}
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

When multiple definitions of the Schottky resistance parameters are found in the parameter file, a priority scheme is used:

1. If a `Region` section contains a `SchottkyResistance` subsection with an `Electrode` specification, then the parameters specified here are chosen.
2. If there is no `Region` section with a `SchottkyResistance` subsection having an `Electrode` specification, then `Material` sections are searched next. If a `Material` section contains a `SchottkyResistance` subsection with an `Electrode` specification, then the parameters specified are chosen.
3. If there is no `Region` section or `Material` section with a `SchottkyResistance` subsection having an `Electrode` specification, only constant parameters can be specified. First, the global `Electrode` sections are searched for constant parameters:

```
Electrode = "gate" {
    # no mole fraction dependency allowed
    SchottkyResistance {
        Rinf = 2.4000e-09 , 5.2000e-09    # [Ohm*cm^2]
    }
}
```

If this section is found, then the parameters here are chosen. If not, a `Region` section with a `SchottkyResistance` subsection and constant parameters is searched for:

```
Region = "reg1" {
    SchottkyResistance {
        Rinf = 2.4000e-09 , 5.2000e-09    # [Ohm*cm^2]
    }
}
```

If a `Region` section is found, then the parameters are chosen here. If not, a `Material` section with a `SchottkyResistance` subsection and constant parameters is the last possible specification:

```
Material = "Silicon" {
    SchottkyResistance {
        Rinf = 2.4000e-09 , 5.2000e-09    # [Ohm*cm^2]
    }
}
```

4. If nothing is specified at all, then the default values are chosen, with their values:

- $m_t = 0.19$ ,  $R_\infty = 2.4 \times 10^{-9} \Omega\text{cm}^2$ ,  $\Phi_B = 0.6 \text{ eV}$  for electrons
- $m_t = 0.19$ ,  $R_\infty = 5.2 \times 10^{-9} \Omega\text{cm}^2$ ,  $\Phi_B = 0.51 \text{ eV}$  for holes [7]

These parameters can be specified independently in different sections as previously described. The priority scheme is applied independently of each of them.

When multiple regions are connected to a Schottky resist contact, the value of the Schottky resistance at the common vertices on the contact is taken as a maximum of the region

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

Schottky resistances. In other words, the Schottky resistance at the common vertices is computed for each region with the region parameters, and the maximum value among regions is chosen.

In the case of alloys, parameters are interpolated from the composing materials. For example, in the case of  $\text{Ge}_x\text{Si}_{1-x}$ , the value of  $R_\infty$  is computed from the two side materials and the corresponding mole-fraction specification. The command file is:

```
Physics (Material = "SiliconGermanium") {    # Gex(x)Si(1-x)
    * Mole fraction material: SiGe(x=0) = Silicon
    * Mole fraction material: SiGe(x=1) = Germanium
    MoleFraction(
        xFraction = 0.35
    )
}
```

The parameter file is:

```
Material = "Silicon" {
    SchottkyResistance {
        Rinf = 2.3000e-09 , 5.1000e-09 # [Ohm*cm^2]
    }
}
Material = "Germanium" {
    SchottkyResistance {
        Rinf = 2.4000e-09 , 5.0000e-09 # [Ohm*cm^2]
    }
}
```

The model is applied to each contact vertex and checks the sign of the doping concentration  $N_{D,0} - N_{A,0}$ . If the sign is positive, the electron parameters are used to compute  $R_d$  for the vertex. If the sign is negative, the hole parameters are used.

As an option, you can specify a constant metal workfunction parameter instead of a mole fraction-dependent Schottky barrier parameter  $\Phi_B$ . In this case, Sentaurus Device computes the Schottky barrier  $\Phi_B$  in [Equation 120](#) as the difference between the specified contact workfunction and the electron affinity of the semiconductor (in the case of n-type semiconductors), or as the difference between the valence band energy and the contact workfunction (in the case of p-type semiconductors). Although only a constant contact workfunction is allowed, the computed Schottky barrier  $\Phi_B$  captures the mole-fraction dependency more physically through the electron affinity and the bandgap mole-fraction dependencies.

To use the metal contact workfunction option to compute the Schottky barrier  $\Phi_B$ , specify the `Workfunction` keyword in the `SchottkyResistance` subsection, of the `Electrode` section, of the parameter file.

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

For example, to compute the Schottky barrier  $\Phi_B$  corresponding to a 4.1 eV workfunction at the contact `top2`, specify in the parameter file:

```
Electrode = "top2" {
    SchottkyResistance {
        # PhiB = 0.6 , 0.51      # [eV]
        Workfunction = 4.1
    }
}
```

In this case, the values of the Schottky barrier  $\Phi_B$  in the parameter file (if specified by `PhiB`) are ignored because they are internally computed.

For convenience, you can also specify the workfunction in the contact section of the region or material:

```
Region = "reg2" {
    SchottkyResistance {
        Workfunction = 4.1
        Xmax(0) = 0.0
        Xmax(1) = 0.1
        Xmax(2) = 0.6
        Electrode = "top" {
            Rinf = 2.5000e-09 , 5.2000e-09      # [Ohm*cm^2]
            Rinf(1) = 2.6000e-09 , 5.2000e-09  # [Ohm*cm^2]
            Rinf(2) = 2.9000e-09 , 5.2000e-09  # [Ohm*cm^2]
        }
    }
}
```

In this case, note that the workfunction is applied to `reg2` only. If the contact extends over multiple regions, the workfunction can be activated only in certain regions by explicitly specifying `Workfunction` in the respective region.

When the workfunction is specified in both the `Electrode` section and the `Region` or `Material` sections, the `Electrode` section takes precedence.

A more general model for Schottky resistance can be implemented using the Schottky resistance PMI (see [Schottky Resistance on page 1434](#)). In this case, Schottky-distributed resistance  $R_d$  can be an arbitrary function of lattice temperature, electron temperature, hole temperature, electron affinity, band gap, bandgap narrowing, conduction-band effective density-of-states, valence-band effective density-of-states, and effective intrinsic density,  $R_d = R_d(T, T_n, T_p, \chi, E_g, E_{bgn}, N_C, N_V, n_{i,eff})$ . The PMI model can access the built-in Schottky resistance model parameters (constant or mole fraction-dependent) using the vertex-based PMI support function `InitModelParameter()` (see [Runtime Support for Vertex-Based PMI Models on page 1237](#)).

---

## Resistive Interfaces

The distributed resistance interface model is supported for metal–semiconductor and metal–metal interfaces only. The interfaces are treated as discontinuous double-point interfaces.

To activate the model, specify the keyword `DistResist` in the `Physics` section of the resistive interface:

```
Physics (MaterialInterface="AlGaAs/Aluminum") {  
    DistResist=1e-3  
}
```

The contribution that the distributed resistance model makes to the device equations is determined by multiplying the current density by the partial contact area (length in two dimensions) assigned to a mesh vertex. An improved approach for calculating the region-interface vertex areas in three dimensions can be activated by specifying `BM_VoronoiRISurfaces3D` in the global `Math` section. This option computes the interface-vertex areas that correspond to the control surfaces for interface vertices and that are consistent with the box method control volumes.

Like Schottky contacts, the behavior of a Schottky interface can be emulated using the doping-dependent resistivity model described by [Equation 120](#). The model is activated for Ohmic metal–semiconductor interfaces by the keyword `DistResist=SchottkyResistance` in the `Physics` section of the interface:

```
Physics(MaterialInterface="AlGaAs/Aluminum") {  
    DistResist=SchottkyResistance  
}
```

The Schottky resistance parameters can be mole dependent or constant. When the parameter file contains multiple definitions of the Schottky resistance parameters, the same priority scheme used for contacts is applied.

As for the contacts, you can specify a constant metal workfunction parameter instead of a mole fraction–dependent Schottky barrier parameter  $\Phi_B$ . The syntax is similar to contacts, but the `Workfunction` keyword must be specified in the `RegionInterface` section or the `MaterialInterface` section instead of the `Electrode` section.

The `DistResist` model is the default for distributed interface resistance simulations. However, to improve convergence performance, an alternative model is available. To activate this alternative model at the metal–semiconductor interface, specify `MSDistResist=<float>` or `MSDistResist=SchottkyResistance` in the `Physics` section, as follows:

```
Physics(MaterialInterface="AlGaAs/Aluminum") {  
    MSDistResist=1e-3  
}
```

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

```
Physics(MaterialInterface="AlGaAs/Aluminum") {  
    MSDistResist=SchottkyResistance  
}
```

A doping-dependent Schottky barrier model is supported for metal–semiconductor Schottky barriers. Image force lowering of the potential energy barrier is included in Schottky contacts. The model is based on the distributed resistance interface model. The Schottky barrier height is dependent on the doping on the semiconductor side.

#### Note:

This model works at the metal–semiconductor interface. It does not work for contacts.

The effects of image force rounding of the potential energy barrier are included. The magnitude of the bias-dependent image force lowering ( $\Delta\phi$ ), relative to the band bending ( $E_b$ ), is defined as [8]:

$$\frac{\Delta\phi}{E_b} = \left[ 16\pi \frac{E_b^{3/2} x_m}{E_{11} \omega^2} \right]^{-1/2} 1 + 3 \frac{x_m}{\omega} \quad (121)$$

$$E_{11} = 9.05 \times 10^{-7} \left[ \frac{N}{K_s K_d^2} \right]^{1/3} \quad (122)$$

$$\frac{x_m}{\omega} = \frac{2}{3} + \frac{1}{3} \cos \frac{\delta}{3} - \frac{1}{\sqrt{3}} \sin \frac{\delta}{3} \quad (123)$$

$$\cos \delta = 1 - \frac{27}{32\pi(E_b/E_{11})^{3/2}} \quad (124)$$

where  $K_s$  is the static dielectric constant,  $K_d$  is the high-frequency dielectric constant,  $E_{11}$  is a constant of the material with the unit of energy, and  $N$  is the electron density.

The syntax can be used in either the `RegionInterface` or `MaterialInterface` section. You can activate the model by specifying the following:

- The keyword `DistResist=SchottkyResistance` activates the Schottky interface resistance model.
- The option `SRDoping` activates the image-force potential lowering model in Schottky contacts.

For example:

```
Physics(MaterialInterface="AlGaAs/Aluminum") {  
    DistResist=SchottkyResistance  
    SRDoping  
}
```

A more general model for interface Schottky resistance can be implemented using the Schottky resistance PMI (see [Schottky Resistance on page 1434](#)). In this case,

## Chapter 10: Boundary Conditions

### Electrical Boundary Conditions

Schottky-distributed resistance  $R_d$  can be an arbitrary function of lattice temperature, electron temperature, hole temperature, electron affinity, band gap, bandgap narrowing, conduction-band effective density-of-states, valence-band effective density-of-states, and effective intrinsic density,  $R_d = R_d(T, T_n, T_p, \chi, E_g, E_{bgn}, N_C, N_V, n_{i,eff})$ . The PMI model can access the built-in Schottky resistance model parameters using the vertex-based PMI support function `InitModelParameter()` (see [Runtime Support for Vertex-Based PMI Models on page 1237](#)).

---

## Boundaries Without Contacts

Outer boundaries of the device that are not contacts are treated with ideal Neumann boundary conditions:

$$\epsilon \nabla \phi + \vec{P} = 0 \quad (125)$$

$$\vec{J}_n \cdot \hat{n} = 0 \quad \vec{J}_p \cdot \hat{n} = 0 \quad (126)$$

[Equation 126](#) also applies to semiconductor–insulator interfaces.

---

## Application Notes

Setting `Barrier` in the `Electrode` statement is typically used to define the semiconductor Schottky barrier, but it also can be used for all other types of contact where it simply shifts the contact electrostatic potential by the specified value. It can be useful for some fitting and calibration, but it might lead to zero bias currents if these barrier values are different for contacts placed in the same region or material.

Given differences in the `Barrier` setting and to avoid zero bias currents, there are special rules for specifying `WorkFunction` in the `Electrode` statement. The `WorkFunction` is ignored for contact mesh nodes if it is located at:

- Only in metals or only in semiconductors with Ohmic boundary conditions
- An interface between a semiconductor and any other type of material with Ohmic boundary conditions at the semiconductor side

Like the `Barrier` setting, the `WorkFunction` is used to define the semiconductor Schottky barrier including contacts placed at metal–semiconductor interfaces. In addition, it is used for contacts placed at metal–insulator interfaces or solely in insulator regions.

## Chapter 10: Boundary Conditions

### Floating Contacts

---

## Floating Contacts

This section discusses floating contacts.

### Floating Metal Contacts

The charge on a floating contact (for example, a floating gate in an EEPROM cell) is specified in the `Electrode` section:

```
Electrode { ...
    { name="FloatGate" charge=1e-15 }
}
```

In the case of a floating metal contact (a contact not in touch with any semiconductor region), the electrostatic potential is determined by solving the Poisson equation with the following charge boundary condition:

$$\underline{\epsilon} \hat{n} \cdot \nabla \phi dS = Q \quad (127)$$

where  $\hat{n}$  is the normal vector on the floating contact surface  $S$ , and  $Q$  is the specified charge on the floating contact. The electrostatic potential on the floating-contact surface is assumed to be constant.

An additional capacitance between floating contact and control contact is necessary if, for example, EEPROM cells are simulated in a 2D approximation, to account for the additional influence on the capacitance from the real 3D layout. The additional capacitance can be specified in the `Electrode` section using the keyword `FGcap`.

For example, if you have `ContGate` and `FloatGate` contacts, additional `ContGate/FloatGate` capacitance is specified as:

```
Electrode {
    { name="ContGate" voltage=10 }
    { name="FloatGate" charge=0 FGcap=(value=3e-15 name="ContGate") }
}
```

where `value` is the capacitance value between `FloatGate` and `ContGate`. For the 1D case, the capacitance unit is  $F/\mu m^2$ ; for the 2D case,  $F/\mu m$ ; and for the 3D case,  $F$ .

If the floating-contact capacitance is specified, [Equation 127](#) changes to:

$$\underline{\epsilon} \hat{n} \cdot \nabla \phi dS + C_{FC}(\phi_{CC} - \phi_{FC} + \phi_B) = Q \quad (128)$$

where:

- $C_{FC}$  is the specified floating-contact capacitance.
- $\phi_{FC}$  is the floating-contact potential.

## Chapter 10: Boundary Conditions

### Floating Contacts

- $\phi_{CC}$  is the potential on the control electrode (specified in the `FGcap` statement).
- $\phi_B$  is a barrier specified in the `Electrode` section for the floating contact, which can be used as a fitting parameter (default value is zero).

#### Note:

If the control contact is a metal, then the value of  $\phi_{CC}$  is defined by the applied voltage and the metal barrier/workfunction (see [Contacts on Insulators on page 260](#)). However, if the control contact touches a semiconductor region, then  $\phi_{CC}$  is equal to the applied voltage with an averaged built-in potential on the contact.

Floating contacts can have several capacitance values for different control contacts. For example:

```
Electrode {
    { name="source" voltage=0 }
    { name="ContGate" voltage=10 }
    { name="FloatGate" charge=0 FGcap( (value=3e-15 name="ContGate")
        (value=2e-15 name="source") )
}
```

In transient simulations, Sentaurus Device takes the charge specified in the `Electrode` section as an initial condition. After each time step, the charge is updated due to tunneling and hot-carrier injection currents. For a description of tunneling and hot-carrier injection, see [Chapter 24 on page 819](#) and [Chapter 25 on page 844](#), respectively.

---

## Floating Semiconductor Contacts

Sentaurus Device can simulate floating semiconductor contacts by connecting an electrode with charge boundary condition to an insulated semiconductor region.

Within that floating-contact region, Sentaurus Device solves the Poisson equation with the following charge boundary condition:

$$[q(p - n + N_D - N_A) + \rho_{\text{trap}}]dV = Q_{\text{FC}} \quad (129)$$

where  $Q_{\text{FC}}$  denotes the total charge on the floating gate and  $\rho_{\text{trap}}$  is the charge density contributed by traps and fixed charges (see [Chapter 17 on page 543](#)). The integral is calculated over all nodes of the floating-contact region [9].

The charge  $Q_{\text{FC}}$  is a boundary condition that must be specified in a Sentaurus Device `Electrode` statement in exactly the same way as for a floating metal contact:

```
Electrode {
    { name="floating_gate" Charge=1e-14 }
}
```

## Chapter 10: Boundary Conditions

### Floating Contacts

It is also possible to use the charge as a goal in a Quasistationary command:

```
Quasistationary (Goal {Name="floating_gate" Charge = 1e-13})  
  { Coupled {Poisson Electron} }
```

Sentaurus Device automatically identifies a floating semiconductor contact based on information in the geometry (.tdr) file. It is not necessary for the charge contact to cover the entire boundary of the floating semiconductor region. A small contact is sufficient if the floating region has the same doping type throughout. However, if both n-type and p-type volumes exist in the floating region, both volumes must be associated with the floating contact defining the floating region. This can be achieved by having a contact in touch with both volumes.

It is assumed that no current flows within the floating region. Therefore, the quasi-Fermi potential for electrons and holes is identical and constant within the floating region:

$$\Phi_n = \Phi_p = \Phi \quad (130)$$

Therefore, the electron and hole densities  $n$  and  $p$  are functions of the electrostatic potential  $\phi$  and the quasi-Fermi potential  $\Phi$  as discussed in [Quasi-Fermi Potential With Boltzmann Statistics on page 231](#) and [Fermi Statistics on page 233](#). Sentaurus Device does not solve the electron and hole continuity equations within a floating region.

In transient simulations, Sentaurus Device takes the charge specified in the `Electrode` section as an initial condition. After each time step, the charge is updated due to hot-carrier injection or tunneling currents including tunneling to traps. For a description of tunneling models and how to enable them, see [Chapter 24 on page 819](#). For hot-carrier injection models, see [Chapter 25 on page 844](#).

The floating-contact capacitance can be specified in the `Electrode` statement in exactly the same way as for the floating metal contact (see [Floating Metal Contacts on page 280](#)). The only difference is that the potential on the semiconductor floating contact is not always a constant. Therefore, defining a constant capacitance might not be strictly correct, and it gives only approximate results. Sentaurus Device uses the semiconductor floating-contact Fermi level to compute the charge associated with the floating-contact capacitance and solves an equilibrium problem (zero voltages and zero charges, see [Equilibrium Solution on page 231](#) for setting numeric parameters to get the equilibrium solution) at the beginning to find a reference Fermi level.

For plotting purposes, floating semiconductor contacts are handled differently from other electrodes. Instead of the voltage, the value of the quasi-Fermi potential  $\Phi$  is displayed.

## Chapter 10: Boundary Conditions

### Thermal Boundary Conditions

---

## Thermal Boundary Conditions

This section discusses thermal boundary conditions.

---

### Boundary Conditions for Lattice Temperature

For the solution of [Equation 70](#), [Equation 71](#), [Equation 72](#), and [Equation 78](#), thermal boundary conditions must be applied. Wachutka [10] is followed, but the difference in thermo-powers between the semiconductor and metal at Ohmic contacts is neglected. For free, thermally insulating, surfaces:

$$\hat{\kappa n} \cdot \nabla T = 0 \quad (131)$$

where  $\hat{n}$  denotes a unit vector in the direction of the outer normal.

At thermally conducting interfaces, thermally resistive (nonhomogeneous Neumann) boundary conditions are imposed:

$$\hat{\kappa n} \cdot \nabla T = \frac{T_{\text{ext}} - T}{R_{\text{th}}} \quad (132)$$

where  $R_{\text{th}}$  is the external thermal resistance, which characterizes the thermal contact between the semiconductor and adjacent material.

For the special case of an ideal heat sink ( $R_{\text{th}} \rightarrow 0$ ), Dirichlet boundary conditions are imposed:

$$T = T_{\text{ext}} \quad (133)$$

By default, the Dirichlet boundary condition is imposed. For the Neumann boundary condition ([Equation 132](#)), you can specify  $R_{\text{th}}$  with the keyword `SurfaceResistance` in the specification of the `Thermode` (see [Table 358 on page 1738](#)). Alternatively, you can use `SurfaceConductance` to specify  $1/R_{\text{th}}$ .

When the lattice temperature equation is solved, the lattice temperature (`Temperature`) and lattice heat flux (`lHeatFlux`) at thermodes are added automatically to the `.plt` file in the corresponding `Thermode` section. The unit for heat flux is W for three dimensions, W/ $\mu\text{m}$  for two dimensions, and W/ $\mu\text{m}^2$  for one dimension.

At insulator–insulator, insulator–semiconductor, semiconductor–semiconductor, metal–semiconductor, metal–metal, and metal–insulator interfaces, a distributed thermal resistance is supported as well.

To activate this feature, specify the distributed resistance by using the `DistrThermalResist` keyword in the `Physics` section of the respective interface.

## Chapter 10: Boundary Conditions

### Periodic Boundary Conditions

The following example activates a distributed thermal resistance at the interface between region "reg1" and region "reg2" with the value 0.001 cm<sup>2</sup>K/W:

```
Physics (RegionInterface="reg1/reg2") {
    DistrThermalResist=1e-3
}
```

As for thermionic emission, double points are used at the interfaces where distributed thermal resistance has been activated. In this case, double points allow a discontinuity in the lattice temperature on the two sides of the interface.

Assuming an interface between materials 1 and 2 such that  $\Delta T = T_2 - T_1 > 0$ , if  $\vec{S}_{L,1}$  is the heat flux density entering material 1 and  $\vec{S}_{L,2}$  is the heat flux density leaving material 2, the thermal boundary condition at the interface between materials 1 and 2 can be written as:

$$\vec{S}_{L,2} = \vec{S}_{L,1} \quad (134)$$

$$\vec{S}_{L,1} = \frac{T_2 - T_1}{R_{d,th}} \quad (135)$$

where  $R_{d,th}$  is the interface-distributed resistance defined with `DistrThermalResist` as described here.

A PMI model for the calculation of the distributed thermal resistance is available. The PMI model allows you to specify the distributed thermal resistance by using physical variables at the two sides of the interface. See [Distributed Thermal Resistance on page 1336](#).

---

## Boundary Conditions for Carrier Temperatures

For the carrier temperatures  $T_n$  and  $T_p$ , at the thermal contacts, fast relaxation to the lattice temperature (boundary condition  $T_n = T_p = T$ ) is assumed.

For other boundaries, adiabatic conditions for carrier temperatures are assumed:

$$\hat{\kappa}_n \hat{n} \cdot \nabla T_n = \hat{\kappa}_p \hat{n} \cdot \nabla T_p = 0 \quad (136)$$

When the hydrodynamic model is used, the lattice temperature (`Temperature`), the lattice heat flux (`lHeatFlux`), and the carrier heat fluxes (`eHeatFlux` and `hHeatFlux`) at thermodes are added automatically to the `.plt` file in the corresponding `Thermode` section. The unit for heat fluxes is W for three dimensions, W/μm for two dimensions, and W/μm<sup>2</sup> for one dimension.

---

## Periodic Boundary Conditions

The use of periodic boundary conditions (PBCs) can be helpful for simulations of periodic device structures, for example, lines or arrays of cells. Instead of building a periodic

## Chapter 10: Boundary Conditions

### Periodic Boundary Conditions

simulation domain, you can supply PBCs at the lower and upper boundary planes (perpendicular to the main axis of the coordinate system) of a single cell. Periodicity of the solution means that the two boundary planes are virtually identified, thereby forming a PBC interface: The solution is (weakly) continuous across this interface, and currents (or fluxes in the general case) can flow from one side to the other.

You can select between two different numeric PBC approaches, referred to as the Robin PBC (RPBC) approach and mortar PBC (MPBC) approach. Both approaches support the following:

- Several transport models, namely, drift-diffusion, thermodynamic, and hydrodynamic transport
- Both conforming and nonconforming geometries and meshes
- Two-dimensional and 3D device structures
- The conservation of the fluxes (for example, current conservation for the continuity equations)
- Both one-fold and two-fold periodicity, that is, PBC can be applied in one axis direction and simultaneously in two directions for 3D structures

---

## Robin PBC Approach

The RPBC approach uses a current (or flux) model between both sides of the PBC interface of the form:

$$\alpha(u_1 - u_2) - j_{12} = 0 \quad (137)$$

where:

- $u_1$  and  $u_2$  are the solution variables of an equation at both sides of the PBC interface.
- $j_{12}$  is the flux density of the equation across the PBC interface.
- $\alpha$  represents a user-provided tuning parameter. The tuning parameter allows you to determine how strongly the local current density across the interface reduces the discontinuity of the solution variable. A large  $\alpha$  reduces the jumps of the solution variable, while for  $\alpha = 0$ , no continuity is enforced at all and no current will flow across the interface.

---

## Mortar PBC Approach

In the MPBC approach, both sides of the PBC interface are effectively glued together on one side (the mortar side), while on the other side (the nonmortar side), a weak continuity condition for the equation potential (mostly, the solution variable) is imposed. Therefore, the

## Chapter 10: Boundary Conditions

### Periodic Boundary Conditions

MPBC approach is not symmetric with respect to the selection of the mortar side if the mesh is nonconforming. The side where the accuracy requirements are larger (and, typically, the mesh is finer) should be selected as the mortar side. The equation potential for the carrier continuity equations are the corresponding quasi-Fermi potentials, while for the other equations, the solution variable is chosen. The MPBC approach does not require a current model across the PBC interface.

---

## Specifying Periodic Boundary Conditions

Periodic boundary conditions are activated by using `PeriodicBC` in the global or device `Math` section. [Table 211 on page 1573](#) provides a complete list of possible options.

## Specifying Robin Periodic Boundary Conditions

RPBCs can be activated individually for all supported equations (see [Electrostatic Potential on page 229](#), [Chapter 8 on page 238](#), and [Chapter 9 on page 245](#)). The solution variable  $u$  of the selected equation (for example, the electrostatic potential  $\phi$  of the Poisson equation), and the corresponding flux density  $j$  are listed in [Table 32](#).

*Table 32 Solution variables and fluxes used in periodic boundary conditions*

$u$	$j$	Description
$\phi$	$\epsilon \nabla \phi$	<a href="#">Electrostatic Potential on page 229</a>
$n$	$\vec{J}_n$	<a href="#">Chapter 8 on page 238</a>
$p$	$\vec{J}_p$	
$T_n$	$\vec{S}_n$	<a href="#">Chapter 9 on page 245</a>
$T_p$	$\vec{S}_p$	
$T$	$\kappa \nabla T$	<a href="#">Chapter 9 on page 245</a>

An RPBC is typically activated by `PeriodicBC(<options>)`, where `<options>` provides a selection of the equations and boundaries. Multiple specifications of `<options>` are possible to select several equations: `PeriodicBC( (<options1>) ... (<optionsN>) )`.

### Note:

If the material parameters or the doping concentration differ on both sides of the interface, RPBC might result in nonphysical solutions.

## Chapter 10: Boundary Conditions

### Periodic Boundary Conditions

An example of specifying RPBCs for different equations and boundaries is:

```
Math {
    PeriodicBC(
        (Direction=0 Coordinates=(-1.0 2.0))
        (Poisson Direction=1 Coordinates=(-1e50 1e50))
        (Electron Direction=1 Coordinates=(-1e50 1e50) Factor=2.e8)
    )
}
```

Here, the first option applies periodic boundary conditions to all equations in the direction of the x-axis and for the coordinates  $-1.0 \mu\text{m}$  and  $2.0 \mu\text{m}$ . The next two options specify periodic boundary conditions for the electron and hole continuity equations in the y-direction and for the device side coordinates. `Factor` determines the tuning parameter  $\alpha$  in the flux density model and defaults to  $1.e8$ .

## Specifying Mortar Periodic Boundary Conditions

Periodic boundary conditions using the MPBC approach are activated in the global or device `Math` section by:

```
PeriodicBC( ( Type=MPBC Direction=1 MortarSide=Ymax ) )
```

`Direction` selects the coordinate axis where the PBC interface is created (that is, here the y-axis). `MortarSide` specifies that the plane at maximum coordinate values is taken as the mortar side (the default is the minimum coordinate plane). The maximum and minimum coordinate planes are extracted automatically. Using `MortarSide` implies the MPBC approach (making `Type` and `Direction` redundant), which means the following specification enables a two-fold MPBC simulation:

```
PeriodicBC( ( MortarSide=Xmin MortarSide=Ymax ) )
```

---

## Application Notes

Note that:

- The RPBC approach supports, in contrast to the MPBC approach, some flexibility with respect to the involved equations.
- The MPBC approach treats the PBC interface essentially as a heterointerface. Conductor regions touching the PBC interface are not supported in MPBC. Heterointerfaces touching the PBC interface are allowed, but there are cases where the periodicity is further relaxed.
- If the device structure forms a heterointerface at the PBC interface, the MPBC approach must be used. The RPBC approach provides nonphysical results.

## Chapter 10: Boundary Conditions

### Discontinuous Interfaces

## Specialized Linear Solver for MPBC

Using MPBC in combination with iterative linear solvers (such as ILS), you might observe some convergence problems. If feasible, using a direct solver (such as PARDISO) should resolve the problem. Otherwise, an improved preconditioner or the use of extended precision improves the convergence behavior in many cases.

There is a specialized linear solver available for MPBC simulations that utilizes the advantages of the iterative solver ILS and improves the robustness of the simulations. This solver can be enabled by a (temporary) user interface. Specifying `UseSchurSolver` in the global `Math` section replaces the `Blocked` method by the specialized MPBC solver for all `Coupled` statements. `Coupled` statements using other methods are not affected. Note that this solver is not yet available for AC analysis.

---

## Discontinuous Interfaces

Discontinuous interfaces generalize the framework used for the simulation of heterointerfaces at semiconductor–semiconductor interfaces (see [Abrupt and Graded Heterojunctions on page 59](#) and [Chapter 26 on page 870](#)). They build the basis for models generating discontinuities of physical quantities across region interfaces.

---

## Representation of Physical Quantities Across Interfaces

Several interface conditions result in discontinuous physical quantities across these interfaces. An interface is a *discontinuous interface* if all physical quantities use a discontinuous representation across this interface. The continuity of selected quantities is guaranteed by applying (often implicitly) corresponding interface conditions. By default, the interface conditions at discontinuous interfaces imply continuity for the solution variable.

You can enforce every interface to be a discontinuous interface by using `Discontinuity` in any `Physics` section. For example:

```
Physics ( MaterialInterface= "Silicon/Oxide" ) {
    Discontinuity * applies to all interfaces of the two materials
}
Physics ( Material= "Silicon" ) {
    Discontinuity * applies to all interfaces to other materials
}
```

Discontinuous interfaces are allowed between any two regions, independent of their material or material group (semiconductor, insulator, or conductor). They are implicitly or explicitly used for the following interface models:

- `Dipole` (see [Dipole Layer on page 230](#))
- `Discontinuity`

## Chapter 10: Boundary Conditions

### References

- DistrThermalResist (see [Boundary Conditions for Lattice Temperature on page 283](#))
- HeteroInterface (see [Abrupt and Graded Heterojunctions on page 59](#))
- Thermionic, eThermionic, hThermionic (see [Chapter 26 on page 870](#))
- Concept of ‘semiconductor floating gate’ (see [Destination of Injected Current on page 845](#))
- Optics standalone simulations (see [Controlling Computation of Optical Problem in Solve Section on page 688](#))

---

## Interface Conditions at Discontinuous Interfaces

At general discontinuous interfaces (that is, involving arbitrary material groups), not all interface conditions are yet supported. So far, the interface conditions related to drift-diffusion, thermodynamic, and hydrodynamic transport are allowed. Simulations using only heterointerfaces (that is, semiconductor–semiconductor interfaces including the `HeteroInterface` flag) as discontinuous interfaces support the full set of transport equations and interface conditions. Note that a discontinuous semiconductor–semiconductor interface must be a heterointerface.

---

## Critical Points

Geometric locations, where several interfaces intersect or an interface intersects the boundary or contact, are so-called critical points. At such critical points, not all interface conditions can be fulfilled, in general, and conflicts are resolved by implicit rules.

Critical points are supported if the set of discontinuous interfaces either is empty, or is the entire set of involved interfaces, or consists of heterointerfaces.

---

## References

- [1] A. Schenk and S. Müller, “Analytical Model of the Metal-Semiconductor Contact for Device Simulation,” in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 5, Vienna, Austria, pp. 441–444, September 1993.
- [2] A. Schenk, *Advanced Physical Models for Silicon Device Simulation*, Wien: Springer, 1998.
- [3] S. M. Sze, *Physics of Semiconductor Devices*, New York: John Wiley & Sons, 2nd ed., 1981.

## Chapter 10: Boundary Conditions

### References

- [4] A. M. Cowley and S. M. Sze, "Surface states and barrier height of metal-semiconductor systems," *Journal of Applied Physics*, vol. 36, no. 10, pp. 3212–3220, 1965.
- [5] W. Mönch, "Chemical trends of barrier heights in metal-semiconductor contacts: on the theory of the slope parameter," *Applied Surface Science*, vol. 92, pp. 367–371, February 1996.
- [6] A. Tugulea and D. Dascalu, "The image-force effect at a metal-semiconductor contact with an interfacial insulator layer," *Journal of Applied Physics*, vol. 56, no. 10, pp. 2823–2831, 1984.
- [7] K. Varahramyan and E. J. Verret, "A Model for Specific Contact Resistance Applicable for Titanium Silicide–Silicon Contacts," *Solid-State Electronics*, vol. 39, no. 11, pp. 1601–1607, 1996.
- [8] V. L. Rideout and C. R. Crowell, "Effects of Image Force and Tunneling on Current Transport in Metal–Semiconductor (Schottky Barrier) Contacts," *Solid-State Electronics*, vol. 13, no. 7, pp. 993–1009, 1970.
- [9] S. Sugino *et al.*, "Analysis of Writing and Erasing Procedure of Flotox EEPROM Using the New Charge Balance Condition (CBC) Model," in *Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits (NUPAD IV)*, Seattle, WA, USA, pp. 65–69, May 1992.
- [10] G. K. Wachutka, "Rigorous Thermodynamic Treatment of Heat Generation and Conduction in Semiconductor Device Modeling," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 11, pp. 1141–1149, 1990.

# 11

## Transport Equations in Metals, Organic Materials, and Disordered Media

---

*This chapter describes transport equations in materials other than low-defect, inorganic semiconductors.*

---

### Singlet Exciton Equation

In organic semiconductor devices, besides the simulation of the electrical part consisting of solving the Poisson and continuity equations, the *singlet exciton equation* is introduced to account for optical properties of these materials related to Frenkel excitons.

This equation takes into account only singlet excitons because triplet excitons do not contribute directly to light emission. The equation models the dynamics of the generation, diffusion, recombination, and radiative decay of singlet excitons in organic semiconductors.

The singlet exciton equation, governing the transport of excitons in organic semiconductors, is given by:

$$\frac{\partial n_{\text{se}}}{\partial t} = R_{\text{bimolec}} + \nabla \cdot D_{\text{se}} \nabla n_{\text{se}} - \frac{n_{\text{se}} - n_{\text{se}}^{\text{eq}}}{\tau} - \frac{n_{\text{se}} - n_{\text{se}}^{\text{eq}}}{\tau_{\text{trap}}} - R_{\text{se}} \quad (138)$$

where:

- $n_{\text{se}}$  is the singlet exciton density.
- $R_{\text{bimolec}}$  is the carrier bimolecular recombination rate acting as a singlet exciton generation term.
- $D_{\text{se}}$  is the singlet exciton diffusion constant.
- $\tau, \tau_{\text{trap}}$  are the singlet exciton lifetimes.
- $R_{\text{se}}$  is the net singlet exciton recombination rate.

The first term on the right-hand side of [Equation 138](#) accounts for the creation of excitons through bimolecular recombination (see [Bimolecular Recombination Model on page 538](#)).

The second term describes exciton diffusion; while radiative decay associated with light emission is accounted for by the third and fourth terms.

Nonradiative exciton destruction and conversion of excitons to free electron and free hole populations are described by the net exciton recombination rate (fifth term):

$$\begin{aligned} R_{\text{se}} &= R_{\text{se}-nf} + R_{\text{se}-pf} \\ R_{\text{se}-nf} &= (v_{\text{se}} + v_n)\sigma_{\text{se}-n} n_{\text{se}} n \\ R_{\text{se}-pf} &= (v_{\text{se}} + v_p)\sigma_{\text{se}-p} n_{\text{se}} p \end{aligned} \quad (139)$$

where:

- $R_{\text{se}-nf}$ ,  $R_{\text{se}-pf}$  are the exciton recombination rate due to free electron and free hole populations, respectively.
- $v_{\text{se}} = v_{\text{se},0}$  is the exciton velocity.
- $v_n = v_{n,0}\sqrt{T/300\text{K}}$  and  $v_p = v_{p,0}\sqrt{T/300\text{K}}$  are the electron and hole velocities.  $v_{\text{se},0}$ ,  $v_{n,0}$ , and  $v_{p,0}$  are set in the parameter file.
- $\sigma_{\text{se}-n}$  and  $\sigma_{\text{se}-p}$  are the reaction cross sections between the singlet exciton, and the electron and hole, respectively.

In addition, the net exciton recombination rate  $R_{\text{se}}$  might contain exciton interface dissociation rates converted to volume terms if the exciton interface dissociation model is activated (see [Exciton Dissociation Model on page 539](#)).

## Boundary and Continuity Conditions for Singlet Exciton Equation

At electrodes and thermodes, equilibrium is assumed for the singlet exciton population:

$$n_{\text{se}}(T) = n_{\text{se}}^{\text{eq}}(T) = \gamma g_{\text{ex}}(N_C(T) + N_V(T)) \exp -\frac{E_{\text{g,eff}} - E_{\text{ex}}}{kT} \quad (140)$$

where  $\gamma = 1/4$ , and  $g_{\text{ex}}$  and  $E_{\text{ex}}$  are the singlet exciton degeneracy factor and binding energy, respectively.

Conditions for the singlet exciton equation at interfaces depend on the interface type. The interface type is dictated by the two regions adjacent to the interface.

For interfaces where the singlet exciton equation is solved only in one of the regions of the interface, the boundary conditions imposed are either [Equation 140](#) (default) or  $\nabla n_{\text{se}} \cdot n = 0$  (zero flux), depending on the user selection.

For a heterointerface where the singlet exciton equation is solved in both regions, the continuity conditions imposed are thermionic-like. In this case, you can select the barrier  $\Delta E$  to be the difference between the band gaps, the conduction bands, or the valence bands of the two regions.

## Chapter 11: Transport Equations in Metals, Organic Materials, and Disordered Media

### Singlet Exciton Equation

Assuming that at heterointerfaces between materials 1 and 2 (that is, regions 1 and 2),  $\Delta E > 0$  ( $E_{g,2} > E_{g,1}$  for example), and  $j_{se,2}$  is the singlet exciton flux leaving material 2 and  $j_{se,1}$  is the singlet exciton flux entering material 1, the interface boundary condition can be written as:

$$\begin{aligned} j_{se,1} &= j_{se,2} \\ j_{se,2} &= v_{si} n_{se,2} - n_{se,1} \exp -\frac{\Delta E}{kT} \end{aligned} \tag{141}$$

where  $n_{se,2}$  and  $n_{se,1}$  are the singlet exciton densities of materials 2 and 1 at the heterointerface, and  $v_{si}$  is the organic heterointerface emission velocity defined in the parameter file.

---

## Using the Singlet Exciton Equation

To activate the singlet exciton equation, the keyword `SingletExciton` must be specified in the `Coupled` statement of the `Solve` section (see [Coupled Statement on page 191](#)). Then, the regions where the equation will be solved are selected by specifying the keyword `SingletExciton` with different options in the `Physics` section of the respective regions (by default, none of the regions is selected for solving the singlet exciton equation, so you must select at least one region).

The singlet exciton generation and recombination terms in [Equation 138](#) are switched off by default. They can be activated regionwise by specifying the corresponding keyword as an option for `Recombination` in the `SingletExciton` section of the respective region `Physics` section (see [Table 260 on page 1650](#)).

The keyword `Bimolecular` activates the bimolecular generation (first term). The keywords `Radiative` and `TrappedRadiative` switch on radiative decay of singlet excitons either directly or trap-assisted (third and fourth terms). Nonradiative exciton destruction (fifth term) is activated by `eQuenching` and `hQuenching`. For example:

```
Physics(Region="EML-ETL") {
    SingletExciton(Recombination(Bimolecular eQuenching hQuenching))
}
```

For interfaces where the singlet exciton equation is solved only in one of the regions of the interface, the default boundary conditions are defined by [Equation 140](#).

To switch to zero flux boundary condition  $\nabla n_{se} \cdot \hat{n} = 0$ , the keyword `FluxBC` must be specified as an option for `SingletExciton` in the interface `Physics` section:

```
Physics(RegionInterface="Region_0/Region_2") {
    SingletExciton(FluxBC)
}
```

For heterointerfaces, the default barrier type for the singlet exciton flux injection across the interface (defined in [Equation 141](#)) is the bandgap energy difference.

## Chapter 11: Transport Equations in Metals, Organic Materials, and Disordered Media

### Singlet Exciton Equation

The barrier can be switched to conduction band or valence band type by specifying the keyword `BarrierType` with the option `CondBand` or `ValBand` in the `SingletExciton` section of the `heterointerface Physics` section:

```
Physics(RegionInterface="Region_0/Region_2") {
    SingletExciton(BarrierType(CondBand))
}
```

Parameters used with the singlet exciton equation must be specified in the `SingletExciton` section of the parameter file.

*Table 33 Parameters for singlet exciton equation*

Symbol	Parameter name	Default value	Location	Unit
$\gamma$	gamma	0.25	region	1
$\tau$	tau	$1 \times 10^{-7}$	region	s
$\tau_{\text{trap}}$	tau_trap	$1 \times 10^{-8}$	region	s
$L_{\text{diff}}$	l_diff	$1 \times 10^{-3}$	region	cm
$D_{\text{se}} = L_{\text{diff}}^2 / \tau$			region	cm <sup>2</sup> /s
$\sigma_{\text{se}-n}$	ex_cXsection	$1 \times 10^{-8}$	region	cm <sup>2</sup>
$\sigma_{\text{se}-p}$	ex_cXsection	$1 \times 10^{-8}$	region	cm <sup>2</sup>
$E_{\text{ex}}$	Eex	0.015	region	eV
$g_{\text{ex}}$	gex	4	region	1
$v_{\text{se},0}$	vth	$1 \times 10^7$	region	cm/s
$v_{n,0}$	vth_car	$1 \times 10^3$	region	cm/s
$v_{p,0}$	vth_car	$1 \times 10^3$	region	cm/s
$v_{\text{si}}$	vel	$1 \times 10^8$	interface	cm/s

---

## Transport in Metals

The simulation of current transport in metals or semi-metals is important for interconnection problems in ICs. The current density in metals is given by:

$$\vec{J}_M = -\sigma(\nabla\Phi_M + P\nabla T) \quad (142)$$

where:

- $\sigma$  is the metal conductivity.
- $\Phi_M$  is the Fermi potential in the metal.
- $P$  is the metal thermoelectric power.
- $T$  is the lattice temperature.

The second term in [Equation 142](#) accounts for the Seebeck effect, and it is nonzero only when computation of metal thermoelectric power is enabled (see [Thermoelectric Power on page 1032](#)). For the steady-state case,  $\nabla \cdot \vec{J}_M = 0$ . Therefore, the equation for the Fermi potential inside of metals is:

$$\nabla \cdot (\sigma(\nabla\Phi_M + P\nabla T)) = 0 \quad (143)$$

The following temperature dependence is applied for  $\rho = 1/\sigma$ :

$$\rho = \rho_0(1 + \alpha_1(T - 273 \text{ K}) + \alpha_2(T - 273 \text{ K})^2) \quad (144)$$

All these resistivity parameters can be specified in the parameter file as:

```
Resistivity {
  * Resist(T) = Resist0*( 1 + TempCoef*(T-273) + TempCoef2*(T-273)^2 )
  Resist0    = <value>      # [Ohm*cm]
  TempCoef   = <value>      # [1/K]
  TempCoef2 = <value>      # [1/K^2]
}
```

No specific keyword is required in the command file because Sentaurus Device recognizes all conductor regions and applies the appropriate equations to these regions and interfaces. The metal conductivity equation [Equation 143](#) is a part of the `Contact` equation, which is solved automatically by default. If the keyword `NoAutomaticCircuitContact` is specified in the `Math` section or only the Poisson equation is solved, the `Contact` equation must be added explicitly in the `Coupled` statement to account for conductivity in metals.

To switch off current transport in metals, specify the keyword `-MetalConductivity` in the `Math` section.

**Note:**

If you use the `-MetalConductivity` option, then all contacts placed directly on metal regions will be electrically disconnected from these regions.

## Electric Boundary Conditions for Metals

The following boundary conditions are used:

- At contacts connected to metal regions, the Dirichlet condition  $\Phi_M = V_{\text{applied}}$  is applied for the Fermi potential.
- Interface conditions always include the displacement current  $\vec{J}_D$  to ensure conservation of current.
- For interfaces between metal and insulator, the equations are:

$$\begin{aligned}\vec{J}_M \cdot \hat{n} &= \vec{J}_D \cdot \hat{n} \\ \phi &= \Phi_M - \Phi_{MS}\end{aligned}\tag{145}$$

where  $\Phi_{MS}$  is the workfunction difference between the metal and an intrinsic semiconductor selected (internally by Sentaurus Device) as a reference material, and  $\hat{n}$  is the unit normal vector to the interface. The electrostatic potential inside metals is computed as  $\phi = \Phi_M - \Phi_{MS}$ .

- At metal–semiconductor interfaces, by default, there is an Ohmic boundary condition. Schottky boundary conditions can be selected as an option.

The Ohmic boundary condition at metal–semiconductor interfaces reads:

$$\begin{aligned}\vec{J}_M \cdot \hat{n} &= (\vec{J}_n + \vec{J}_p + \vec{J}_D) \cdot \hat{n} \\ \phi &= \Phi_M + \phi_0 \\ n &= n_0 \\ p &= p_0\end{aligned}\tag{146}$$

where:

- $\phi_0$  is the equilibrium electrostatic potential (the built-in potential).
- $n_0$  and  $p_0$  are the electron and hole equilibrium concentrations (see [Ohmic Contacts on page 258](#)).

There are several options for the Schottky interface (refer to the equations in [Schottky Contacts on page 261](#)). The potential barrier between metal and semiconductor is computed automatically, and the barrier tunneling model (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 826](#)) and the barrier lowering model (see [Barrier Lowering at Schottky Contacts on page 265](#)) can be applied.

To select these models, use the following syntax in the command file:

```
Physics(MaterialInterface = "Metal/Silicon") { Schottky
    eRecVelocity=1e6 hRecVelocity=1e6 }
```

## Chapter 11: Transport Equations in Metals, Organic Materials, and Disordered Media

### Transport in Metals

```
Physics(MaterialInterface = "Metal/Silicon") { Schottky  
    BarrierLowering }  
Physics(MaterialInterface = "Metal/Silicon") { Schottky }
```

The default values of electron and hole surface recombination velocities are  $eRecVel=2.573e6$  and  $hRecVel=1.93e6$ , respectively. Similarly to `Electrode` conditions, if `Schottky` is not specified but the `SurfaceSRH` keyword is specified, then Sentaurus Device will apply Ohmic boundary conditions to the electrostatic potential and the surface recombination for both electron and hole carrier current densities on such interfaces.

#### Note:

At nonmetal interfaces, the keyword `SurfaceSRH` activates a different model from the one described in this section (see [Surface SRH Recombination Model on page 486](#)).

Both Ohmic and Schottky interfaces support a distributed resistance model similar to the one described in [Resistive Contacts on page 270](#). If a distributed resistance is specified at the interface, a distributed voltage drop  $\Delta\phi(R_d) = R_d(J_n + J_p + J_D) \cdot n$  is applied to the existing boundary condition for potential, where  $\Delta\phi(R_d)$  is the voltage drop across the interface, and  $R_d$  is the distributed resistance at the interface. In addition, for Ohmic interfaces, emulation of a Schottky interface using a doping-dependent resistivity model is possible. These options are activated as following:

```
# Distributed resistance at Ohmic interface  
Physics(MaterialInterface = "Metal/Silicon") {DistResist=1e-6}  
  
# Distributed resistance at Schottky interface  
Physics(MaterialInterface = "Metal/Silicon") {Schottky DistResist=1e-6}  
  
# Distributed resistance at Ohmic interface emulating  
# a Schottky interface  
Physics(MaterialInterface = "Metal/Silicon") {DistResist=SchottkyResist}
```

For Schottky interfaces, the distributed resistance model works with all other options previously described.

For all other metal boundaries, the Neumann condition  $\vec{J}_M \cdot \hat{n} = 0$  is applied.

#### Note:

By default, Sentaurus Device computes output currents through a contact using integration over the associated wells. For metal contacts, the integration can be switched off and the local current can be used instead by specifying the keyword `DirectCurrent` in the `Math` section. This option might produce a wrong current at the metal contact.

## Metal Workfunction

To specify the metal workfunction, use the `Bandgap` parameter set. For example:

```
Material = "Gold" {
    Bandgap { WorkFunction = 5 # [eV] }
}
```

It is also possible to account for workfunction variations within the metal.

If such variations can be characterized by mole-fraction variations and if the metal is treated as a mole fraction-dependent material (see [Mole-Fraction Materials on page 64](#)), then `WorkFunction` in the `Bandgap` parameter set can be specified as a mole fraction-dependent quantity. For example:

```
Material = "Metal" {
    Bandgap {
        Xmax(0)=0.0
        Xmax(1)=0.2
        WorkFunction(0) = 4.53# [eV]
        WorkFunction(1) = 4.38# [eV]
    }
}
```

The positional dependency of the workfunction in metals also can be specified directly in the `Physics` section of the command file using one of three different methods:

```
Physics (Material = "<name>" | Region = "<name>") {
    MetalWorkfunction (
        [ Workfunction=<wf> ] |
        [ Workfunction=(<wf1>, <x1>, <y1>, <z1>)
        Workfunction=(<wf2>, <x2>, <y2>, <z2>)
        ...
        Workfunction=(<wfN>, <xN>, <yN>, <zN>) ] |
        [ SFactor= "<dataset_name-or-pmi_model_name>" [Factor=<scale>]
        [Offset=<offset>]]
    )
}
```

The first method (`Workfunction=<wf>`) simply specifies a constant workfunction value for the material or region that overrides any specification from the parameter file.

The next method allows the specification of an arbitrary number of workfunction-position quadruplets. The workfunction at a vertex in the metal is assigned the value of the workfunction from the quadruplet whose position is closest to the vertex.

Alternatively, you can use the `SFactor` parameter to specify a dataset name (which includes the PMI user fields `PMIUserField0` through `PMIUserField299`) from which the vertex workfunction values will be taken, or a space factor PMI model can be specified that calculates the vertex workfunction values directly (see [Space Factor on page 1479](#)). If `Factor=<scale>` is specified, then the `SFactor` values are normalized and then are

multiplied by this factor. If `Offset=<offset>` is specified, then this value is added to the raw or scaled `SFactor` values.

## Metal Workfunction Randomization

The workfunction in metal regions can be randomized using specifications in the command file:

```
Physics (Material = "<name>" | Region = "<name>") {
    MetalWorkfunction (
        Randomize (
            AverageGrainSize=<size>                      # [micrometers]
            GrainProbability=(<P1>, <P2>, ...)
            GrainWorkfunction=(<wf1>, <wf2>, ...)      # [eV]
            RandomSeed=<seed>
            AtInsulatorInterface
            UniformDistribution
        )
    )
}
```

The approach taken assumes that the metal consists of randomized grains of varying size and shape that can be characterized with an average grain size. It also assumes that the grains in the metal occur with a finite number of orientations, and the number of grains for each orientation can be characterized with a probability of occurrence. All grains of the same orientation are assumed to have the same workfunction, but the workfunction can be different for each orientation.

`AverageGrainSize` is used to determine the average number of grains for a region, which then is used as the expectation value for a Poisson distribution, random number generator to determine the actual number of grains for the region. Each grain can have a different shape and size.

You can specify an arbitrary number of `GrainProbability` values, but their sum must be equal to one. In addition, there must be a one-to-one correspondence between `GrainProbability` values and `GrainWorkfunction` values.

By default, the seed for the random number generator used in the randomization process is different for every simulation. However, `RandomSeed` can be specified if required. This allows a particular randomization to be repeated if the same seed is used in a subsequent simulation of the same device on the same computer.

The `AtInsulatorInterface` option confines the randomization to the metal–insulator interface vertices. However, the resulting workfunction values at the interface will be exactly the same as those obtained when the entire metal region is randomized.

The `UniformDistribution` option *attempts* to use grains with a uniform size that are distributed uniformly. The success of this is highly dependent on the grid used in the metal region.

The workfunction in metal regions can be saved in a plot file for visualization by specifying `MetalWorkfunction` in the `Plot` section of the command file.

---

## Temperature in Metals

In metals, only the lattice temperature is defined. If the lattice temperature is solved, in metals, it obeys the following equation:

$$c_L \frac{\partial T}{\partial t} - \nabla \cdot \kappa \nabla T = -\nabla \psi_M \cdot \vec{J}_M \quad (147)$$

where  $\kappa$  is the thermal conductivity (see [Thermal Conductivity on page 1019](#)). If current transport in metals is switched off, then the right-hand side of [Equation 147](#) vanishes. However, a heat flow in metals can still be simulated.

---

## Conductive Insulators

Leakage currents in dielectrics can be modeled by allowing dielectrics to have conductive properties. The conductive insulator (leaky insulator) model implements dielectrics that have nonzero (but typically low) conductivity (a metal-like property), besides the pure dielectric properties.

In a conductive insulator, the electrostatic potential is computed using the Poisson equation, similar to a pure dielectric. As in metals, conductivity is modeled by the Fermi potential (solving [Equation 143](#)) and then the leakage current is computed using [Equation 142](#). Since the model does not allow net charge in the conductive insulator, the Poisson equation remains unchanged by adding conductive properties to the insulator.

The equation for the Fermi potential inside a conductive insulator is the same used for metals (see [Equation 143](#)), where  $\sigma$  is now the conductivity of the conductive insulator and  $\Phi_M = \Phi_{CI}$  is the Fermi potential in the conductive insulator.

The following boundary conditions are used for this equation:

- At contacts connected to conductive insulator regions, the Dirichlet condition  $\Phi_{CI} = V_{\text{applied}}$  is applied for the Fermi potential.
- Interface conditions always include the displacement current  $\vec{J}_D$  in insulators, conductive insulators, and semiconductors to ensure conservation of current.
- For conductive insulator–semiconductor interfaces, Ohmic-like boundary conditions, thermionic-like boundary conditions, and boundary conditions for floating regions are available.

Leakage from or to a semiconductor to or from a conductive insulator is mainly determined by two factors: the conductive properties of the interface and the bulk resistivity of the

conductive insulator. The bulk resistivity-limited conduction corresponds to the case when the current flow is limited by the high resistivity of the conductive insulator region, and the interface has zero resistivity.

This case is modeled by the Ohmic-like boundary conditions:

$$\begin{aligned}\vec{J}_{\text{CI}} \cdot \hat{n} &= (\alpha \vec{J}_n + (1 - \alpha) \vec{J}_p) \cdot \hat{n} \\ \Phi_{\text{CI}} &= \phi - \phi_0\end{aligned}\quad (148)$$

where:

- $\Phi_{\text{CI}}$  and  $\vec{J}_{\text{CI}}$  are the Fermi potential and current density on the interface.
- $\phi$  is the electrostatic potential (the solution of the Poisson equation in insulator and semiconductor regions).
- $\phi_0$  is the built-in potential.
- $\hat{n}$  is the unit normal vector to the interface.
- $\vec{J}_n$  and  $\vec{J}_p$  are the electron and hole currents on the semiconductor side of the interface.
- $\alpha$  is the fraction ( $0 \leq \alpha \leq 1$ ) of the current density of the conductive insulator going to the electron current density.

Thermionic-like emission at the interface models the case where conduction is limited by both interface properties and the bulk resistivity of the conductive insulator. Thermionic boundary conditions read:

$$\begin{aligned}\vec{J}_{\text{CI}} \cdot \hat{n} &= (\vec{J}_{n,\text{th}} + \vec{J}_{p,\text{th}}) \cdot \hat{n} \\ \vec{J}_{n,\text{th}} &= aq \left[ v_n(T_n)n - v_n(T) \frac{T}{T_n}^{1.5} N_C \exp \frac{E_F^{\text{CI}} - E_C}{kT} \right] \\ \vec{J}_{p,\text{th}} &= aq \left[ v_p(T_p)p - v_p(T) \frac{T}{T_p}^{1.5} N_V \exp \frac{E_V - E_F^{\text{CI}}}{kT} \right]\end{aligned}\quad (149)$$

where:

- $\vec{J}_{n,\text{th}}$  and  $\vec{J}_{p,\text{th}}$  are the electron and hole thermionic-emission current densities on the interface from the semiconductor side.
- $v_n(T) = (kT/2\pi m_n)^{0.5}$  and  $v_p(T) = (kT/2\pi m_p)^{0.5}$  are ‘emission velocities’ at the interface.
- $T$ ,  $T_n$ , and  $T_p$  are the lattice, electron, and hole temperatures, respectively.

## Chapter 11: Transport Equations in Metals, Organic Materials, and Disordered Media

### Conductive Insulators

- $E_F^{\text{CI}}$ ,  $E_C$ , and  $E_V$  are the conductive insulator Fermi-level energy, conduction band energy, and valence band energy, respectively.
- $a$  is a constant set through the parameter file, which has the default value of 2.

By using a conductive insulator layer between a semiconductor floating region and a semiconductor region, leakage from a semiconductor floating-gate can be emulated. The boundary condition used in this case is:

$$\vec{J}_{\text{CI}} \cdot \hat{n} = -\sigma \nabla \Phi \quad (150)$$

where  $\Phi$  is the Fermi potential close to the interface. In this case, during transient simulations, the charge update on the floating gate is computed as  $J_{\text{CI}} \Delta t$ , where  $J_{\text{CI}}$  is the total current at the conductive insulator–floating gate interface, and  $\Delta t$  is the current time step. The activation of this type of boundary condition is automatic if the boundary condition at the conductive insulator–semiconductor floating-gate interface is Ohmic and the simulation is transient (to allow for charging and discharging of the floating gate).

The model is activated by using the keyword `CondInsulator` in the `Physics` section to make the dielectric conductive and by adding the `CondInsulator` equation in the `Solve` section to actually solve the Fermi potential:

```
Physics(Region="insulator") {
    ...
    CondInsulator
    ...
}

Solve {
    ...
    Coupled {Poisson Electron Hole Contact CondInsulator}
    ...
}
```

The parameters for the resistivity of conductive insulators (see [Equation 144](#)) are specified in the `Resistivity` parameter set:

```
Material = "Si3N4" {
    Resistivity {
        * Resist(T) = Resist0 * (1 + TempCoef * (T-273))
        Resist0 = 3.0e9                                # [Ohm*cm]
        TempCoef = 43.0e-4                            # [1/K]
    }
}
```

Boundary conditions according to [Equation 148](#) are used by default. The parameter  $\alpha$  in [Equation 148](#) is given by the value of `curw` in the `CondInsCurr` parameter set:

```
MaterialInterface = "Si3N4/AlGaN" {
    CondInsCurr {
        curw = 1.0      # [1]
```

## Chapter 11: Transport Equations in Metals, Organic Materials, and Disordered Media

### Conductive Insulators

```
    }  
}
```

Thermionic emission at the semiconductor–conductive insulator interface is enabled by adding eThermionic or hThermionic in the Physics section:

```
Physics(MaterialInterface="Si3N4/AlGaN") {  
    *----- Insulator/AlGaN -----  
    ...  
    eThermionic  
    hThermionic  
    ...  
}
```

where the parameter  $a$  (see [Equation 149](#)) is specified in the parameter file:

```
MaterialInterface = "Si3N4/AlGaN" {  
    ThermionicEmission {  
        A = 2, 2  
    }  
}
```

For nonthermionic interfaces, an equilibrium solution is needed internally to solve the CondInsulator equation. The equilibrium solution is computed automatically by a nonlinear iteration. Sentaurus Device allows you to specify numeric parameters of the nonlinear iteration as options to the keyword EquilibriumSolution in the Math section (see [Equilibrium Solution on page 231](#) and [Table 217 on page 1605](#)). The following example forces the Newton solver to solve up to 100 iterations:

```
Math {  
    ...  
    EquilibriumSolution(Iterations=100)  
    ...  
}
```

# 12

## Semiconductor Band Structure

---

*This chapter discusses the relevance of band structure to the simulation of semiconductor devices.*

For device simulation, the most fundamental property of a semiconductor is its band structure. Realistic band structures are complex and can be fully accounted for only in Monte Carlo simulations (see the *Sentaurus™ Device Monte Carlo User Guide*). In Sentaurus Device, the band structure is simplified to the following quantities:

- Energies of the conduction and valence band edges or, parameterized differently, the band gap and electron affinity (see [Band Gap and Electron Affinity on page 305](#))
- Density-of-states (DOS) masses for electrons and holes, or parameterized differently, the band edge DOS (see [Effective Masses and Effective Density-of-States on page 319](#))

---

### Intrinsic Density

The band gap and band edge DOS are summarized in the intrinsic density  $n_i(T)$  (for undoped semiconductors):

$$n_i(T) = \sqrt{N_C(T)N_V(T)} \exp\left(-\frac{E_g(T)}{2kT}\right) \quad (151)$$

and the effective intrinsic density (including doping-dependent bandgap narrowing):

$$n_{i,\text{eff}} = n_i \exp\left(\frac{E_{\text{bgn}}}{2kT}\right) \quad (152)$$

In devices that contain different materials, the electron affinity  $\chi$  (see [Band Gap and Electron Affinity](#)) is also important. Along with the band gap, it determines the alignment of conduction and valence bands at material interfaces.

## Band Gap and Electron Affinity

The band gap is the difference between the lowest energy in the conduction band and the highest energy in the valence band. The electron affinity is the difference between the lowest energy in the conduction band and the vacuum level.

---

### Selecting the Bandgap Model

Sentaurus Device supports different bandgap models: `BennettWilson` (default), `deAlamo`, `JainRoulston`, `OldSlotboom`, `Slotboom` (the same as `OldSlotboom` with different default parameters), and `TableBGN`.

You specify the bandgap model in the `EffectiveIntrinsicDensity` statement in the `Physics` section. The following example activates the `Slotboom` model:

```
Physics {  
    EffectiveIntrinsicDensity(BandGapNarrowing (Slotboom))  
}
```

By default, bandgap narrowing is active. Bandgap narrowing can be switched off with the keyword `NoBandGapNarrowing`:

```
Physics {  
    EffectiveIntrinsicDensity(NoBandGapNarrowing)  
}
```

#### Note:

To plot the band gap including the bandgap narrowing, specify `EffectiveBandGap` in the `Plot` section of the command file. The quantity that Sentaurus Device plots when `BandGap` is specified does not include bandgap narrowing.

[Table 36 on page 318](#) and [Table 37 on page 319](#) list the model parameters available for calibration (see [Bandgap and Electron-Affinity Models](#)).

---

## Bandgap and Electron-Affinity Models

Sentaurus Device models the lattice temperature-dependency of the band gap as [1]:

$$E_g(T) = E_g(0) - \frac{\alpha T^2}{T + \beta} \quad (153)$$

where  $E_g(0)$  is the bandgap energy at 0 K, and  $\alpha$  and  $\beta$  are material parameters (see [Table 36](#)).

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

To allow  $E_g(0)$  to differ for different bandgap models, it is written as:

$$E_g(0) = E_{g,0} + \delta E_{g,0} \quad (154)$$

$E_{g,0}$  is an adjustable parameter common to all models. For the TableBGN and JainRoulston models,  $\delta E_{g,0} = 0$ . Each of the other models offers its own adjustable parameter for  $\delta E_{g,0}$  (see [Table 36 on page 318](#)).

The effective band gap results from the band gap reduced by bandgap narrowing:

$$E_{g,\text{eff}}(T) = E_g(T) - E_{\text{bgn}} \quad (155)$$

The electron affinity  $\chi$  is the energy separation between the conduction band and the vacuum. For BennettWilson, delAlamo, OldSlotboom, Slotboom, and TableBGN, the affinity is temperature dependent and is affected by bandgap narrowing:

$$\chi(T) = \chi_0 + \frac{(\alpha + \alpha_2)T^2}{2(T + \beta + \beta_2)} + \text{Bgn2Chi} \cdot E_{\text{bgn}} \quad (156)$$

where  $\chi_0$  and Bgn2Chi are adjustable parameters (see [Table 36](#)). Bgn2Chi defaults to 0.5 and, therefore, bandgap narrowing splits equally between conduction and valence bands.

For the JainRoulston model, the electron affinity depends also on the doping concentration (especially at high dopings) because bandgap narrowing is doping dependent:

$$\chi(T, N_{A,0}, N_{D,0}) = \chi_0 + \frac{(\alpha + \alpha_2)T^2}{2(T + \beta + \beta_2)} + E_{\text{bgn}}^{\text{cond}} \quad (157)$$

where  $E_{\text{bgn}}^{\text{cond}}$  is the shift in the conduction band due to the JainRoulston bandgap narrowing and  $\chi_0$  is an adjustable parameter as previously defined.

The main difference of the bandgap models is how they handle bandgap narrowing. Bandgap narrowing in Sentaurus Device has the form:

$$E_{\text{bgn}} = \Delta E_g^0 + \Delta E_g^{\text{Fermi}} \quad (158)$$

where  $\Delta E_g^0$  is determined by the particular bandgap narrowing model used, and  $\Delta E_g^{\text{Fermi}}$  is an optional correction to account for carrier statistics (see [Equation 172 on page 316](#)).

## Bandgap Narrowing for Bennett–Wilson Model

Bandgap narrowing for the Bennett–Wilson [2] model (keyword `BennettWilson`) in Sentaurus Device reads:

$$\Delta E_g^0 = \begin{cases} -E_{\text{ref}} \left[ \ln \frac{N_{\text{tot}}}{N_{\text{ref}}} \right]^2 & N_{\text{tot}} \geq N_{\text{ref}} \\ 0 & \text{otherwise} \end{cases} \quad (159)$$

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

The model was developed from absorption and luminescence data of heavily doped n-type materials. The material parameters  $E_{\text{ref}}$  and  $N_{\text{ref}}$  are accessible in the `Bennett` parameter set in the parameter file of Sentaurus Device (see [Table 37 on page 319](#)).

## Bandgap Narrowing for Slotboom Model

Bandgap narrowing for the Slotboom model (keyword `Slotboom` or `OldSlotboom`) (the only difference is in the parameters) in Sentaurus Device reads:

$$\Delta E_g^0 = E_{\text{ref}} \left[ \ln \frac{N_{\text{tot}}}{N_{\text{ref}}} + \sqrt{\ln \frac{N_{\text{tot}}}{N_{\text{ref}}}^2 + 0.5} \right] \quad (160)$$

The models are based on measurements of  $\mu_n n_i^2$  in n-p-n transistors (or  $\mu_p n_i^2$  in p-n-p transistors) with different base doping concentrations and a 1D model for the collector current [3]–[6]. The material parameters  $E_{\text{ref}}$  and  $N_{\text{ref}}$  are accessible in the `Slotboom` and `OldSlotboom` parameter sets in the parameter file (see [Table 37](#)).

## Bandgap Narrowing for del Alamo Model

Bandgap narrowing for the del Alamo model (keyword `delAlamo`) in Sentaurus Device reads:

$$\Delta E_g^0 = \begin{cases} E_{\text{ref}} \ln \frac{N_{\text{tot}}}{N_{\text{ref}}} & N_{\text{tot}} \geq N_{\text{ref}} \\ 0 & \text{otherwise} \end{cases} \quad (161)$$

This model was proposed [7]–[11] for n-type materials. The material parameters  $E_{\text{ref}}$  and  $N_{\text{ref}}$  are accessible in the `delAlamo` parameter set in the parameter file (see [Table 37](#)).

## Bandgap Narrowing for Jain–Roulston Model

Bandgap narrowing for the Jain–Roulston model (keyword `JainRoulston`) in Sentaurus Device is implemented based on the literature [12] and is given by:

$$\Delta E_g^0 = A \cdot N_{\text{tot}}^{1/3} + B \cdot N_{\text{tot}}^{1/4} + C \cdot N_{\text{tot}}^{1/2} + D \cdot N_{\text{tot}}^{1/2} \quad (162)$$

where  $A$ ,  $B$ ,  $C$ , and  $D$  are material-dependent coefficients that can be specified in the parameter file.

The coefficients  $A$ ,  $B$ ,  $C$ , and  $D$  are derived from the quantities defined and described in the literature [12]:

$$A = 1.83 \frac{\Lambda}{N_b^{1/3}} \frac{aR}{\frac{3}{4\pi}} \quad [\text{eV} \cdot \text{cm}] \quad (163)$$

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

$$B = \frac{0.95Ra^{3/4}}{\frac{3}{4\pi}^{1/4}} \quad [\text{eV} \cdot \text{cm}^{3/4}] \quad (164)$$

$$C = \frac{1.57Ra^{3/2}}{N_b \frac{3}{4\pi}^{1/2}} \quad [\text{eV} \cdot \text{cm}^{3/2}] \quad (165)$$

$$D = \frac{1.57R_{(\text{mino})}a^{3/2}}{N_b \frac{3}{4\pi}^{1/2}} \quad [\text{eV} \cdot \text{cm}^{3/2}] \quad (166)$$

where  $a$  is the effective Bohr radius,  $R$  is the Rydberg energy,  $N_b$  is the number of valleys in the conduction or valence band,  $R_{(\text{mino})}$  is the Rydberg energy for the minority carrier band, and  $\Lambda$  is a correction factor.

In general, papers on bandgap narrowing define the Jain–Roulston bandgap narrowing as  $\Delta E_g^0 = C_1 \cdot N_{\text{tot}}^{1/3} + C_2 \cdot N_{\text{tot}}^{1/4} + C_3 \cdot N_{\text{tot}}^{1/2}$ . Comparing this definition to [Equation 162](#), the coefficients  $C_1$ ,  $C_2$ , and  $C_3$  can be mapped to the ones of Sentaurus Device:  $C_1 = A$ ,  $C_2 = B$ , and  $C_3 = C + D$ .

If, on the other hand,  $C_1$ ,  $C_2$ , and  $C_3$  are given, then computing the corresponding Sentaurus Device coefficients  $A$ ,  $B$ ,  $C$ , and  $D$  requires splitting  $C_3$  into  $C$  and  $D$  using [Equation 165](#) and [Equation 166](#).

Sentaurus Device needs four coefficients instead of three to compute  $E_{\text{bgn}}^{\text{cond}}$  and the doping-dependent affinity ([Equation 157](#)) internally.

The expression of  $E_{\text{bgn}}^{\text{cond}}$  is given by:

$$E_{\text{bgn}}^{\text{cond}} = \begin{cases} A \cdot N_{\text{tot}}^{1/3} + C \cdot N_{\text{tot}}^{1/2} & N_{\text{D},0} > N_{\text{A},0} \\ B \cdot N_{\text{tot}}^{1/4} + D \cdot N_{\text{tot}}^{1/2} & \text{otherwise} \end{cases} \quad (167)$$

The coefficients for SiC are available in Sentaurus Device. The default coefficients for SiC are for the material 4H-SiC. You need to define the coefficients for the material 6H-SiC in the parameter file. These coefficients are taken from the literature [\[13\]](#).

The coefficients  $A$ ,  $B$ ,  $C$ , and  $D$  are specified in the `JainRoulston` section of the parameter file:

```
Material = "Silicon" {
    JainRoulston {
        * n-type
        A_n = 1.02e-8      # [eV cm]
        B_n = 4.15e-7      # [eV cm^(3/4)]
        C_n = 1.45e-12     # [eV cm^(3/2)]
        D_n = 1.48e-12     # [eV cm^(3/2)]
        * p-type
        A_p = 1.11e-8      # [eV cm]
```

## Chapter 12: Semiconductor Band Structure

## Band Gap and Electron Affinity

```

        B_p = 4.79e-7      # [eV cm^(3/4)]
        C_p = 3.23e-12     # [eV cm^(3/2)]
        D_p = 1.81e-12     # [eV cm^(3/2)]
    }
}
```

**Table 34** Coefficients and their default values for Jain–Roulston bandgap narrowing model

Symbol	Parameter name	Default value for material						Unit
		Si	Ge	GaAs	4H-SiC	6H-SiC	All others	
<i>A</i>	<i>A_n</i>	1.02e-8	7.30e-9	1.65e-8	1.50e-8	1.12e-8	0.0	eVcm
	<i>A_p</i>	1.11e-8	8.21e-9	9.77e-9	1.30e-8	1.30e-8	0.0	eVcm
<i>B</i>	<i>B_n</i>	4.15e-7	2.57e-7	2.38e-7	6.01e-7	6.67e-7	0.0	eVcm <sup>3/4</sup>
	<i>B_p</i>	4.79e-7	2.91e-7	3.87e-7	4.96e-7	5.50e-7	0.0	eVcm <sup>3/4</sup>
<i>C</i>	<i>C_n</i>	1.45e-12	2.29e-12	1.83e-11	2.93e-12	1.01e-12	0.0	eVcm <sup>3/2</sup>
	<i>C_p</i>	3.23e-12	3.58e-12	3.41e-12	1.15e-12	1.14e-12	0.0	eVcm <sup>3/2</sup>
<i>D</i>	<i>D_n</i>	1.48e-12	2.03e-12	7.25e-11	8.74e-12	1.73e-12	0.0	eVcm <sup>3/2</sup>
	<i>D_p</i>	1.81e-12	2.19e-12	4.84e-13	3.87e-13	6.64e-13	0.0	eVcm <sup>3/2</sup>

## Table Specification of Bandgap Narrowing

You can specify bandgap narrowing using a table, which can be defined in the TableBGN parameter set. This table gives the value of bandgap narrowing as a function of donor or acceptor concentration, or total concentration (the sum of acceptor and donor concentrations).

When specifying acceptor and donor concentrations, the total bandgap narrowing is the sum of the contributions of the two dopant types. If only acceptor or only donor entries are present in the table, the bandgap narrowing contribution for the missing dopant type vanishes. Total concentration and donor or acceptor concentration must not be specified in the same table.

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

Each table entry is a line that specifies a concentration type (Donor, Acceptor, or Total) with a concentration in  $\text{cm}^{-3}$ , and the bandgap narrowing for this concentration in eV. The actual bandgap narrowing contribution for each concentration type is interpolated from the table, using a scheme that is piecewise linear in the logarithm of the concentration.

For concentrations below (or above) the range covered by table entries, the bandgap narrowing of the entry for the smallest (or greatest) concentration is assumed. For example:

```
TableBGN {  
    Total 1e16, 0  
    Total 1e20, 0.02  
}
```

means that for total doping concentrations below  $10^{16} \text{ cm}^{-3}$ , the bandgap narrowing vanishes, then increases up to 20 meV at  $10^{20} \text{ cm}^{-3}$  concentration and maintains this value for even greater concentrations. The interpolation is such that, in this example, the bandgap narrowing at  $10^{18} \text{ cm}^{-3}$  is 10 meV.

Tabulated default parameters for bandgap narrowing are available only for GaAs. For all other materials, you can specify the tabulated data in the parameter file.

#### Note:

You cannot specify mole fraction-dependent bandgap narrowing tables. In particular, Sentaurus Device does not relate the parameters for ternary compound semiconductor materials to those of the related binary materials.

## Schenk Bandgap Narrowing Model

BennettWilson, delAlamo, JainRoulston, OldSlotboom, Slotboom, and TableBGN are all doping-induced bandgap narrowing models. They do not depend on carrier concentrations. High carrier concentrations produced in optical excitation or high electric field injection can also cause bandgap narrowing. This effect is referred to as plasma-induced bandgap narrowing.

The Schenk bandgap narrowing model described in [14] also takes into account the plasma-induced narrowing effect in silicon. In this model, the bandgap narrowing is the sum of two parts:

- An exchange-correlation part, which is a function of plasma density and temperature
- An ionic part, which is a function of activated doping concentration, plasma density, and temperature

Sentaurus Device supports two versions of the Schenk model: a simplified version where bandgap narrowing is computed at  $T = 0 \text{ K}$  and does not depend on temperature, and the full model where temperature dependence is accounted for. You can switch from the simplified model (the default) to the full model by setting `IsSimplified=-1` in the Schenk bandgap narrowing section of the parameter file.

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

The full-model exchange correlation and ionic terms are described by:

$$\Delta_a^{xc}(n, p, T) = -\frac{(4\pi)^3 n_\Sigma^2 \left[ \frac{48n_a}{\pi g_a}^{1/3} + c_a \ln(1 + d_a n_p^{p_a}) \right] + \frac{8\pi\alpha_a}{g_a} n_a \Upsilon^2 + \sqrt{8\pi n_\Sigma} \Upsilon^{5/2}}{(4\pi)^3 n_\Sigma^2 + \Upsilon^3 + b_a \sqrt{n_\Sigma} \Upsilon^2 + 40n_\Sigma^{3/2} \Upsilon} \quad (168)$$

$$\Delta_a^i(N_{tot}, n, p, T) = -\frac{N_{tot}[1 + U^i(n_\Sigma, \Upsilon)]}{\sqrt{\Upsilon n_\Sigma / (2\pi)} [1 + h_a \ln(1 + \sqrt{n_\Sigma} / \Upsilon) + j_a U^i(n_\Sigma, \Upsilon) n_p^{3/4} (1 + k_a n_p^{q_a})]} \quad (169)$$

and, for the simplified model by:

$$\Delta_a^{xc}(n, p, T=0) = -\left[ \frac{48n_a}{\pi g_a}^{1/3} + c_a \ln(1 + d_a n_p^{p_a}) \right] \quad (170)$$

$$\Delta_a^i(N_{tot}, n, p, T=0) = -N_{tot} \frac{0.799\alpha_a}{N_p^{3/4}} \quad (171)$$

where  $a = e, h$  is the index for the carrier type,  $\Upsilon = kT/Ry_{ex}$ ,  $n_\Sigma = n + p$ ,  $N_p = \alpha_e N_D + \alpha_h N_A$ , and  $U^i(n_\Sigma, \Upsilon) = n_\Sigma^2 / \Upsilon^3$ . The default values (silicon) and units for the parameters that can be changed in Sentaurus Device in this model are listed in [Table 35 on page 313](#).

Sentaurus Device implements the Schenk bandgap narrowing model using the framework of the density gradient model (see [Density Gradient Model on page 362](#)). If you want to suppress the quantization corrections contained in this model, then set the density gradient model parameter  $\gamma = 0$  (see [Equation 235 on page 362](#)). Alternatively, you can specify the LocalModelOnly option in the eQuantumPotential and hQuantumPotential statements in the command file. The option LocalModelOnly automatically sets  $\gamma = 0$  for all regions where both LocalModel and LocalModelOnly are invoked. For example:

```
Physics(Region = "Region1_Si") {
    ...
    eQuantumPotential(LocalModel=SchenkBGN_elec LocalModelOnly)
    hQuantumPotential(LocalModel=SchenkBGN_hole LocalModelOnly)
    ...
}
```

The Schenk model is switched on separately for electrons (conduction band correction) and holes (valence band correction). The complete bandgap narrowing correction is the sum of conduction and valence band corrections, and it can be obtained by switching on the Schenk model for both electrons and holes.

First, in the Physics section of silicon regions, SchenkBGN\_elec must be specified as the LocalModel (that is, the model for  $\Lambda_{PMI}$  in [Equation 235 on page 362](#)) for eQuantumPotential (conduction bandgap correction), and SchenkBGN\_hole must be specified as the LocalModel for hQuantumPotential (valence band correction).

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

In addition, because the Schenk model is a bandgap narrowing model itself, all other models must be switched off by specifying the keyword `NoBandGapNarrowing` as the argument for `EffectiveIntrinsicDensity`:

```
Physics(Region = "Region1_Si") {
    ...
    EffectiveIntrinsicDensity (NoBandGapNarrowing)
    eQuantumPotential(LocalModel=SchenkBGN_elec)
    hQuantumPotential(LocalModel=SchenkBGN_hole)
    ...
}
```

Apart from switching on quantum corrections in the `Physics` section, the equations for quantum corrections must be solved to compute the corrections. This is performed by specifying `eQuantumPotential` (for electrons) or `hQuantumPotential` (for holes) or both in the `Solve` section:

```
Solve {
    Coupled (LineSearchDamping=0.01){Poisson eQuantumPotential}
    Coupled {Poisson eQuantumPotential hQuantumPotential}
    quasistationary (
        Goal {name="base" voltage=0.4}
        Goal {name="collector" voltage=2.0}
        Initialstep=0.1 Maxstep=0.1 Minstep=1e-6
    )
    {coupled {poisson electron hole eQuantumPotential
              hQuantumPotential}}
}
```

Finally, to ignore the quantization corrections by the density gradient model, in the parameter file,  $\gamma$  is set to zero. Then, the Schenk model parameter `IsSimplified` is set to 1 if the simplified model ( $T = 0$  K) is used or -1 for the full model (temperature dependent):

```
Material = "Silicon" {
    ...
    *turn off everything in density gradient model except
    *apparent band-edge shift
    QuantumPotentialParameters {gamma = 0 , 0}
    ...
    SchenkBGN_elec {
        ...
        * Selects simplified model (1) or complete Schenk model (-1)
        IsSimplified = 1
    }
    SchenkBGN_hole {
        ...
        * Selects simplified model (1) or complete Schenk model (-1)
        IsSimplified = 1
    }
}
```

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

The Schenk bandgap narrowing model is specifically for silicon. Sentaurus Device permits the model parameters to be changed in the parameter file (the `SchenkBGN_elec` and `SchenkBGN_hole` sections) as an option for also using this bandgap narrowing model for other materials. A full description of the parameters is given in the literature [14] and their default values are summarized in [Table 35](#).

The Schenk bandgap narrowing can be visualized by specifying the data entry `eSchenkBGN` or `hSchenkBGN` or both in the `Plot` section of the command file.

*Table 35 Default parameters for Schenk bandgap narrowing model*

Symbol	Parameter name	Default value (silicon)	Unit
<b>SchenkBGN_elec</b>			
$\alpha_e$	alpha_e	0.5187	1
$g_e$	g_e	12	1
$Ry_{ex}$	Ry_ex	16.55	meV
$a_{ex}$	a_ex	3.719e-7	cm
$b_e$	b_e	8	1
$c_e$	c_e	1.3346	1
$d_e$	d_e	0.893	1
$p_e$	p_e	0.2333	1
$h_e$	h_e	3.91	1
$j_e$	j_e	2.8585	1
$k_e$	k_e	0.012	1
$q_e$	q_e	0.75	1
<b>SchenkBGN_hole</b>			
$\alpha_h$	alpha_h	0.4813	1

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

Table 35 Default parameters for Schenk bandgap narrowing model (Continued)

Symbol	Parameter name	Default value (silicon)	Unit
$g_h$	g_h	4	1
$Ry_{ex}$	Ry_ex	16.55	meV
$a_{ex}$	a_ex	3.719e-7	cm
$b_h$	b_h	1	1
$c_h$	c_h	1.2365	1
$d_h$	d_h	1.1530	1
$p_h$	p_h	0.2333	1
$h_h$	h_h	4.20	1
$j_h$	j_h	2.9307	1
$k_h$	k_h	0.19	1
$q_h$	q_h	0.25	1

## Bandgap Narrowing With Incomplete Ionization

The bandgap narrowing models introduced in [Equation 159](#), [Equation 160](#), [Equation 161](#), and [Equation 162](#), BennettWilson, delAlamo, JainRoulston, OldSlotboom, Slotboom, and TableBGN, use carrier concentrations  $N_{tot}$  by default. Incomplete ionization affects bandgap narrowing. For details about the incomplete ionization model, see [Chapter 13](#) on [page 339](#).

Sentaurus Device implements the bandgap narrowing model with incomplete ionized doping concentration using the framework of the density gradient model (see [Density Gradient Model on page 362](#)). If you want to suppress the quantization corrections contained in this model, then set the density gradient model parameter  $\gamma = 0$  (see [Equation 235](#) on [page 362](#)).

The bandgap narrowing model with incomplete ionization is a correction to the existing bandgap narrowing model. The carrier concentration  $N_{tot}$  is replaced by the incomplete

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

ionized doping concentration. Both the bandgap narrowing model and the incomplete ionization model should be activated within the `Physics` section.

The complete bandgap narrowing correction is the sum of conduction and valence band corrections, and it can be obtained by switching on the bandgap narrowing model for both electrons and holes. In the `Physics` section, you must specify `IncompleteBGN_elec` as the `LocalModel` for `eQuantumPotential` (conduction band correction) and `IncompleteBGN_hole` as the `LocalModel` for `hQuantumPotential` (valence band correction). For example:

```
Physics(Region = "Region1_Si") {  
    ...  
    EffectiveIntrinsicDensity (BandGapNarrowing(model_name))  
    eQuantumPotential(LocalModel=IncompleteBGN_elec)  
    hQuantumPotential(LocalModel=IncompleteBGN_hole)  
    IncompleteIonization(options)  
    ...  
}
```

Since the bandgap narrowing model with incomplete ionization is a correction through the quantum equations, the equations of the quantum potential must be solved to compute the corrections. This correction is activated by specifying `eQuantumPotential` (for electrons) and `hQuantumPotential` (for holes) in the `Solve` section:

```
Solve {  
    Coupled {Poisson eQuantumPotential hQuantumPotential}  
    Quasistationary (  
        Goal {name="base" voltage=0.4}  
        Goal {name="collector" voltage=2.0}  
        Initialstep=0.1 Maxstep=0.1 Minstep=1e-6  
    )  
    {coupled {poisson electron hole eQuantumPotential  
    hQuantumPotential}}  
}
```

## Bandgap Narrowing With Fermi Statistics

Parameters for bandgap narrowing are often extracted from experimental data assuming Maxwell–Boltzmann statistics. However, in the high-doping regime for which bandgap narrowing is important, Maxwell–Boltzmann statistics differs significantly from the more realistic Fermi statistics.

The bandgap narrowing parameters are, therefore, systematically affected by using the wrong statistics to interpret the experiment. For use in simulations that do not use Fermi statistics, this ‘error’ in the parameters is desirable, as it partially compensates the error by using the ‘wrong’ statistics in the simulation. However, for simulations using Fermi statistics, this compensation does not occur.

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

Therefore, Sentaurus Device can apply a correction to the bandgap narrowing to reduce the errors introduced by using Maxwell–Boltzmann statistics for the interpretation of experiments on bandgap narrowing (see [Equation 158 on page 306](#)):

$$\Delta E_g^{\text{Fermi}} = k300K \left[ \ln \frac{N_V N_C}{N_{A,0} N_{D,0}} + F_{1/2}^{-1} \frac{N_{A,0}}{N_V} + F_{1/2}^{-1} \frac{N_{D,0}}{N_C} \right] \quad (172)$$

where the right-hand side is evaluated at 300 K.

By default, correction [Equation 172](#) is switched on for simulations using Fermi statistics and switched off (that is,  $\Delta E_g^{\text{Fermi}} = 0$ ) for simulations using Maxwell–Boltzmann statistics. To switch off the correction in simulations using Fermi statistics, specify

`EffectiveIntrinsicDensity(NoFermi)` in the `Physics` section of the command file. This is recommended if you use parameters for bandgap narrowing that have been extracted assuming Fermi statistics. Sometimes for III–IV materials, the correction [Equation 172](#) is too large, and it is recommended to switch it off in these cases. For Maxwell–Boltzmann statistics, either `EffectiveIntrinsicDensity(NoFermi)` or

`EffectiveIntrinsicDensity()` means that the correction of [Equation 172](#) is set to zero.

Sometimes, the combined effect of mechanical stresses, Fermi correction, and bandgap narrowing results in a very small effective bandgap (~0 eV), particularly in high-doped source/drain regions of MOS devices. In some cases, the low effective bandgap can result in either convergence issues or other numeric issues. Sentaurus Device prints a warning message in the log file for one such vertex in the device if the effective bandgap is below 5% of the unperturbed bandgap (without a bandgap narrowing correction) of the semiconductor material. If you observe such a warning message, then it is recommended either to check the doping or stress at these vertices or to set the `EgMin` parameter (see [Table 36](#)) in the parameter file for the relevant semiconductor material.

## Band Gap and Parameters of Non-Three-Dimensional Density-of-States

All non-3D DOS options available in Sentaurus Device are described in [Non-Three-Dimensional Density-of-States on page 330](#). To set the graphene nanoribbon (GNR) DOS and the carbon nanotube (CNT) DOS, the Fermi velocity should be defined (see [Graphene Nanoribbon Density-of-States on page 331](#) and [Carbon Nanotube Density-of-States on page 331](#)). In addition, the following analytic expression for the CNT band gap can be used as well:

$$E_g = \frac{4\hbar v_F}{3d} \quad (173)$$

where  $v_F$  is the Fermi velocity and  $d$  is the CNT diameter.

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

To activate [Equation 173](#) for the bandgap model, you should use `Formula=2` with, for example, the following `Bandgap` section in the parameter file:

```
Bandgap {  
    Tpar = 300  
    Formula = 2  
    FermiVelocity = 9e7 # [cm/s]  
    CNTdiameter = 4 # [nm]  
}
```

#### Note:

Even if `Formula=2` ([Equation 173](#)) is not used, the parameters `FermiVelocity` and `CNTdiameter` should be used for the CNT DOS and `FermiVelocity` for the GNR DOS in the `Bandgap` section of the corresponding material or region.

For all such non-3D DOS options, you should define the correct settings for the bandgap narrowing and temperature dependence models (see [Selecting the Bandgap Model on page 305](#) and [Bandgap and Electron-Affinity Models on page 305](#)).

---

## Bandgap Parameters

The band gap  $E_{g,0}$  and values of  $\delta E_{g,0}$  for each model are accessible in the `BandGap` section of the parameter file, in addition to the electron affinity  $\chi_0$  and the temperature coefficients  $\alpha$  and  $\beta$ . As an extension to what [Equation 153](#) and [Equation 156](#) suggest, the parameters  $E_{g,0}$  and  $\chi_0$  can be specified at any reference temperature  $T_{\text{par}}$ . By default,  $T_{\text{par}} = 0 \text{ K}$ .

To prevent abnormally low (or negative) band gap, the calculated value of the band gap  $E_g$  (which might include bandgap narrowing and stress effects) is limited if  $E_g < E_{g,\min} + \delta E_{g,\min}$ :

$$E_{g,\text{limited}} = E_{g,\min} + \delta E_{g,\min} \exp[(E_g - E_{g,\min} - \delta E_{g,\min})/\delta E_{g,\min}] \quad (174)$$

## Chapter 12: Semiconductor Band Structure

### Band Gap and Electron Affinity

*Table 36 Bandgap models: Parameters and their default values for silicon*

Symbol	Parameter name	Default	Unit	Band gap at 0 K	Reference
				$E_{g,0} + \delta E_{g,0} + \frac{\alpha T_{\text{par}}^2}{\beta + T_{\text{par}}}$	
$E_{g,0}$	Eg0	1.1696	eV	–	<a href="#">Equation 154</a>
$\delta E_{g,0}$	dEg0(Bennett)	0.0	eV	1.1696	<a href="#">Equation 154</a>
	dEg0(Slotboom)	$-4.795 \times 10^{-3}$	eV	1.1648	
	dEg0(OldSlotboom)	$-1.595 \times 10^{-2}$	eV	1.1537	
	dEg0(delAlamo)	$-1.407 \times 10^{-2}$	eV	1.1556	
$\alpha$	alpha	$4.73 \times 10^{-4}$	eV/K	–	<a href="#">Equation 153</a> , <a href="#">Equation 156</a>
$\alpha_2$	alpha2	0	eV/K	–	<a href="#">Equation 156</a>
$\beta$	beta	636	K	–	<a href="#">Equation 153</a> , <a href="#">Equation 156</a>
$\beta_2$	beta2	0	K	–	<a href="#">Equation 156</a>
$\chi_0$	Chi0	4.05	eV	–	<a href="#">Equation 156</a>
Bgn2Chi	Bgn2Chi	0.5	1	–	<a href="#">Equation 156</a>
$T_{\text{par}}$	Tpar	0	K	–	
$E_{g,\text{min}}$	EgMin	0	eV	–	<a href="#">Equation 174</a>
$\delta E_{g,\text{min}}$	dEgMin	0.01	eV	–	<a href="#">Equation 174</a>

[Table 37](#) summarizes the silicon default parameters for the analytic bandgap narrowing models available in Sentaurus Device.

*Table 37 Bandgap narrowing models: Parameters and their default values for silicon*

Symbol	Parameter	Bennett	Slotboom	Old Slotboom	del Alamo	Unit
$E_{\text{ref}}$	$E_{\text{bgn}}$	$6.84 \times 10^{-3}$	$6.92 \times 10^{-3}$	$9.0 \times 10^{-3}$	$18.7 \times 10^{-3}$	eV
$N_{\text{ref}}$	$N_{\text{ref}}$	$3.162 \times 10^{18}$	$1.3 \times 10^{17}$	$1.0 \times 10^{17}$	$7.0 \times 10^{17}$	$\text{cm}^{-3}$

## Effective Masses and Effective Density-of-States

Sentaurus Device provides two options for computing carrier effective masses and densities of states. The first method, selected by specifying `Formula=1` in the parameter file, computes an effective DOS as a function of carrier effective mass. The effective mass can be either independent of temperature or a function of the temperature-dependent band gap. The latter is the most appropriate model for carriers in silicon and is the default for simulations of silicon devices.

In the second method, selected by specifying `Formula=2` in the parameter file the effective carrier mass is computed as a function of a temperature-dependent DOS. The default for simulations of GaAs devices is `Formula=2`.

---

## Electron Effective Mass and Density-of-States

This section describes the different options.

### Formula=1

The lattice temperature-dependence of the DOS effective mass of electrons is modeled by:

$$m_n = 6^{2/3} (m_t^2 m_l)^{1/3} + m_m \quad (175)$$

where the temperature-dependent effective mass component  $m_t(T)$  is best described in silicon by the temperature dependence of the energy gap [15]:

$$\frac{m_t(T)}{m_0} = a \frac{E_g(0)}{E_g(T)} \quad (176)$$

The coefficient  $a$  and the mass  $m_l$  are defined in the parameter file with the default values provided in [Table 38 on page 320](#). The parameter  $m_m$ , which defaults to zero, allows  $m_n$  to be defined as a temperature-independent quantity if required.

## Chapter 12: Semiconductor Band Structure

### Effective Masses and Effective Density-of-States

The effective DOS in the conduction band  $N_C$  follows from:

$$N_C(m_n, T_n) = 2.5094 \times 10^{19} \frac{m_n^{\frac{3}{2}} T_n^{\frac{3}{2}}}{m_0^{\frac{1}{2}} 300 \text{ K}} \text{ cm}^{-3} \quad (177)$$

### Formula=2

If `Formula=2` is specified in the parameter file, the value for the DOS is computed from  $N_C(300 \text{ K})$ , which is read from the parameter file:

$$N_C(T_n) = N_C(300 \text{ K}) \frac{T_n^{\frac{3}{2}}}{300 \text{ K}} \quad (178)$$

and the electron effective mass is simply a function of  $N_C(300 \text{ K})$ :

$$\frac{m_n}{m_0} = \frac{N_C(300 \text{ K})^{\frac{2}{3}}}{2.5094 \times 10^{19} \text{ cm}^{-3}} \quad (179)$$

## Electron Effective Mass and Conduction Band DOS Parameters

[Table 38](#) lists the default coefficients for the electron effective mass and conduction band DOS models. The values can be modified in the `eDOSMass` parameter set.

### Note:

The default setting for the `Formula` parameter depends on the materials, for example, it is equal to 1 for silicon and 2 for GaAs.

*Table 38 Default coefficients for effective electron mass and DOS models*

Option	Symbol	Parameter name	Electrons	Unit
Formula=1	$a$	<code>a</code>	0.1905	1
	$m_l$	<code>ml</code>	0.9163	1
	$m_m$	<code>mm</code>	0	1
Formula=2	$N_C(300 \text{ K})$	<code>Nc300</code>	$2.890 \times 10^{19}$	$\text{cm}^{-3}$

## Hole Effective Mass and Density-of-States

This section discusses the different options.

### Formula=1

For the DOS effective mass of holes, the best fit in silicon is provided by the expression [16]:

$$\frac{m_p(T)}{m_0} = \frac{\frac{a + bT + cT^2 + dT^3 + eT^4}{1 + fT + gT^2 + hT^3 + iT^4}}{m_m}^{\frac{2}{3}} \quad (180)$$

where the coefficients are listed in [Table 39 on page 322](#). The parameter  $m_m$ , which defaults to zero, allows  $m_p$  to be defined as a temperature-independent quantity. The effective DOS for holes  $N_V$  follows from:

$$N_V(m_p, T_p) = 2.5094 \times 10^{19} \frac{m_p^{\frac{3}{2}} T_p^{\frac{3}{2}}}{m_0^{\frac{3}{2}} 300 \text{ K}} \text{ cm}^{-3} \quad (181)$$

### Formula=2

If `Formula=2` in the parameter file, the temperature-dependent DOS is computed from  $N_V(300 \text{ K})$  as given in the parameter file:

$$N_V(T_p) = N_V(300 \text{ K}) \frac{T_p^{\frac{3}{2}}}{300 \text{ K}} \quad (182)$$

and the effective hole mass is given by:

$$\frac{m_p}{m_0} = \frac{N_V(300 \text{ K})}{2.5094 \times 10^{19} \text{ cm}^{-3}}^{\frac{2}{3}} \quad (183)$$

## Hole Effective Mass and Valence Band DOS Parameters

The model coefficients for the hole effective mass and valence band DOS can be modified in the parameter set `hDOSMass`. [Table 39](#) lists the default parameter values.

### Note:

The default setting for the `Formula` parameter depends on the materials, for example, it is equal to 1 for silicon and it is equal to 2 for GaAs.

**Table 39 Default coefficients for hole effective mass and DOS models**

Option	Symbol	Parameter name	Holes	Unit
Formula=1	a	a	0.4435870	1
	b	b	0.3609528×10 <sup>-2</sup>	K <sup>-1</sup>
	c	c	0.1173515×10 <sup>-3</sup>	K <sup>-2</sup>
	d	d	0.1263218×10 <sup>-5</sup>	K <sup>-3</sup>
	e	e	0.3025581×10 <sup>-8</sup>	K <sup>-4</sup>
	f	f	0.4683382×10 <sup>-2</sup>	K <sup>-1</sup>
	g	g	0.2286895×10 <sup>-3</sup>	K <sup>-2</sup>
	h	h	0.7469271×10 <sup>-6</sup>	K <sup>-3</sup>
	i	i	0.1727481×10 <sup>-8</sup>	K <sup>-4</sup>
	m <sub>m</sub>	mm	0	1
Formula=2	N <sub>V</sub> (300 K)	Nv300	3.140×10 <sup>19</sup>	cm <sup>-3</sup>

## Gaussian Density-of-States for Organic Semiconductors

Gaussian DOS has been introduced to better represent electron and hole effective DOS in disordered organic semiconductors. Within the Gaussian disorder model, electron and hole DOS are represented by:

$$\Gamma(E) = \frac{N_t}{\sqrt{2\pi}\sigma_{\text{DOS}}} \exp -\frac{(E-E_0)^2}{2\sigma_{\text{DOS}}^2} \quad (184)$$

where  $N_t$  is the total number of hopping sites with  $N_t = N_{\text{LUMO}}$  for electrons (LUMO is the lowest unoccupied molecular orbital) and  $N_t = N_{\text{HOMO}}$  for holes (HOMO is the highest occupied molecular orbital),  $E_0$  and  $\sigma_{\text{DOS}}$  are the energy center and the width of the DOS distribution, respectively.

## Chapter 12: Semiconductor Band Structure

### Gaussian Density-of-States for Organic Semiconductors

Electron and hole densities in the most general case (Fermi–Dirac statistics) then are computed using Gaussian distribution from [Equation 184](#) as:

$$\frac{n}{N_{\text{LUMO}}} = \frac{1}{\sqrt{2\pi}\sigma_n} \int_{-\infty}^{\infty} \frac{(E - E_{0n})^2}{2\sigma_n^2} \frac{1}{1 + e^{(E - E_{F,n})/(kT)}} dE = G_n(\zeta_n; \hat{s}_n) \quad (185)$$

$$\frac{p}{N_{\text{HOMO}}} = \frac{1}{\sqrt{2\pi}\sigma_p} \int_{-\infty}^{\infty} \frac{(E - E_{0p})^2}{2\sigma_p^2} \frac{1}{1 + e^{(E_{F,p} - E)/(kT)}} dE = G_p(\zeta_p; \hat{s}_p) \quad (186)$$

where:

- $\zeta_n = \frac{E_{F,n} - E_{0n}}{kT}$ ,  $\zeta_p = \frac{E_{0p} - E_{F,p}}{kT}$

- $\hat{s}_n = \frac{\sigma_n}{kT}$ ,  $\hat{s}_p = \frac{\sigma_p}{kT}$

Sentaurus Device computes Gauss–Fermi integrals  $G_n(\zeta_n; \hat{s}_n)$  and  $G_p(\zeta_p; \hat{s}_p)$  using an analytic approximation [17]. The approximation covers both the nondegenerate and degenerate cases:

$$G_c(\zeta_c; \hat{s}_c) = \begin{cases} \frac{\exp \frac{\hat{s}_c^2}{2} + \zeta_c}{1 + \exp(K(\hat{s}_c)(\zeta_c + \hat{s}_c^2))} & \zeta_c \leq -\hat{s}_c^2 \text{ (nondegenerate region)} \\ \frac{1}{2} \operatorname{erfc} \frac{-\zeta_c}{\hat{s}_c \sqrt{2}} H(\hat{s}_c) & \zeta_c > -\hat{s}_c^2 \text{ (degenerate region)} \end{cases} \quad (187)$$

where the index  $c$  is  $n$  or  $p$ , and the analytic functions  $H$  and  $K$  are given by:

$$H(s) = \frac{\sqrt{2}}{s} \operatorname{erfc}^{-1} \exp -\frac{s^2}{2} \quad (188)$$

$$K(s) = 2 \left[ 1 - \frac{H(s)}{s} \right] \sqrt{\frac{2}{\pi}} \exp \left[ \frac{1}{2} s^2 (1 - H^2(s)) \right] \quad (189)$$

The model is activated regionwise, materialwise, or globally by specifying the keyword `GaussianDOS_Full` in the respective `Physics` section together with the `Fermi` keyword in the global `Physics` section:

```
Physics(Region="Organic_seml") {
    GaussianDOS_full
}

Physics {
    Fermi
}
```

## Chapter 12: Semiconductor Band Structure

### Gaussian Density-of-States for Organic Semiconductors

The model parameters used in [Equation 185](#) and [Equation 186](#) are listed in [Table 40](#). They can be specified in the GaussianDOS\_full section of the parameter file.

*Table 40 Gaussian DOS model parameters*

Parameter symbol	Parameter name	Default value	Unit
$N_t$	$N_{\text{LUMO}}$ (electron)	$1 \times 10^{21}$	$\text{cm}^{-3}$
	$N_{\text{HOMO}}$ (hole)	$1 \times 10^{21}$	
$\sigma_{\text{DOS}}$	$\sigma_n$	0.052	eV
	$\sigma_p$	0.052	
$E_{0n} = E_0 - E_C$ (electron)	$E0$	0.1	eV
$E_{0p} = E_V - E_0$ (hole)		0.1	

In the case of nondegenerate semiconductors (Maxwell–Boltzmann approximation can be used for carrier densities), a simplified version based on a correction of standard densities-of-states  $N_C$  and  $N_V$  is available as well. The densities-of-states are corrected so that equivalent carrier densities are obtained when Gaussian densities-of-states described by [Equation 184](#) are used instead of standard densities-of-states.

Introducing the notations:

$$\begin{aligned} A_e &= \frac{\sigma_n^2}{2k^2} & B_e &= \frac{E_{0n} - E_C}{k} & C_e &= \frac{E_{0n} - E_C}{\sqrt{2}\sigma_n} \\ A_h &= \frac{\sigma_p^2}{2k^2} & B_h &= \frac{E_V - E_{0p}}{k} & C_h &= \frac{E_V - E_{0p}}{\sqrt{2}\sigma_p} \end{aligned} \quad (190)$$

the electron and hole effective densities-of-states are computed in the Maxwell–Boltzmann approximation as:

$$N_C(T) = \frac{N_{\text{LUMO}}}{2} \exp \left( \frac{A_e}{T^2} - \frac{B_e}{T} \right) \operatorname{erfc} \left( \frac{\sqrt{A_e}}{T} \right) - C_e \quad (191)$$

and:

$$N_V(T) = \frac{N_{\text{HOMO}}}{2} \exp \left( \frac{A_h}{T^2} - \frac{B_h}{T} \right) \operatorname{erfc} \left( \frac{\sqrt{A_h}}{T} \right) - C_h \quad (192)$$

The simplified model can be activated regionwise or globally by specifying the keyword GaussianDOS for EffectiveMass in the Physics section of the command file:

```
Physics(Region="Organic_seml") {
    EffectiveMass(GaussianDOS)
}
```

## Chapter 12: Semiconductor Band Structure

### Band Tails

The model parameters used in [Equation 184](#) and [Equation 190](#) and their default values are summarized in [Table 41](#).

*Table 41 Simplified Gaussian DOS model parameters*

Parameter symbol	Parameter name	Default value	Unit
$N_t$	$N_{\text{LUMO}}$ (electron) $N_{\text{HOMO}}$ (hole)	$\text{N\_LUMO}$ $\text{N\_HOMO}$	$1 \times 10^{21}$ $1 \times 10^{21}$
$\sigma_{\text{DOS}}$	$\sigma_n$ $\sigma_p$	$\text{sigmaDOS\_e}$ $\text{sigmaDOS\_h}$	0.052 0.052
$E_{0_{\text{cn}}} = E_0 - E_C$ $E_{0_{\text{v}}} = E_V - E_0$	$\text{E0\_c}$ $\text{E0\_v}$	0.1 0.1	eV

## Band Tails

According to [\[18\]](#), the semiconductor density-of-states in the presence of random dopant fluctuations and crystal defects should have band tails in the band gap. Here, presented in the semiconductor DOS, it is assumed that band tails bring additional mobile carriers into the conduction and valence bands. The DOS band tails might affect subthreshold transport properties of MOSFET applications strongly at cryogenic temperatures.

Gaussian and exponential band tail DOS are accounted for as an extension of both the conduction and valence bands in the following form:

$$D_g(\epsilon) = \frac{N_t}{\sigma \sqrt{2\pi}} \exp\left[-0.5 \left|\frac{\epsilon - \epsilon_0}{\sigma}\right|^2\right] \quad (193)$$

$$D_e(\epsilon) = \frac{N_t}{2\sigma} \exp\left[-\left|\frac{\epsilon - \epsilon_0}{\sigma}\right|\right]$$

where  $N_t$  is a maximum of the band tail concentration,  $\epsilon_0$  is an energy position of the maximum in a reference to the conduction band or the valence band with positive  $\epsilon_0$  shifting the maximum into the band gap, and  $\sigma$  is a characteristic energy width of the band tail distribution.

## Chapter 12: Semiconductor Band Structure

### Band Tails

Based on the band tail DOS of [Equation 193](#), the total electron concentration versus the quasi-Fermi energy  $E_{F,n}$  and temperature  $T$  is computed as the following (similar applies to holes):

$$n(E_{F,n}, T) = n_B(E_{F,n}, T) + n_{bt}(E_{F,n}, T)$$

$$n_{bt}(E_{F,n}, T) = \sum_{-\infty}^{\infty} \frac{D_{g,e}(\epsilon)}{1 + \exp[\frac{\epsilon + E_C - E_{F,n}}{kT}]} d\epsilon \quad (194)$$

where  $n_B(E_{F,n}, T)$  is the electron concentration computed using the DOS and band structure without the band tails, and  $n_{bt}(E_{F,n}, T)$  is the concentration of electrons that occupy the band tail DOS.

To activate the band tail DOS model, you can use the `BandTailDOS`, `eBandTailDOS`, or `hBandTailDOS` statement in the `Physics` section. These statements should be set with one of the following options, as for example, for electron band tails:

```
eBandTailDOS(Exponent)
eBandTailDOS(Gaussian)
```

The band tail DOS parameters of [Equation 193](#) ( $N_t$ ,  $\sigma$ , and  $\epsilon_0$ ) should be set for electron and hole band tails in the following section of the parameter file:

```
BandTailDOS{
    Nt = 1e19, 1e19           # [cm-3]
    sigmaDOS = 0.0075, 0.0075 # [eV]
    E0 = 0.0, 0.0             # [eV]
}
```

By default, Sentaurus Device uses the Fermi–Dirac distribution function in the band tail part of [Equation 194](#) independently of the carrier statistics defined in the `Physics` section. In addition, for better convergence at cryogenic temperatures, Sentaurus Device uses an abrupt distribution function, which simplifies integration in [Equation 194](#).

#### Note:

To have a true distribution function with numeric integration of band tail carriers (with a Gauss–Laguerre quadrature approximation) in [Equation 194](#), use `-SimpleBandTailIntegration` in the `Math` section of the command file.

To have the distribution function in [Equation 194](#) set by the carrier statistics defined in the `Physics` section, use `-FermiBandTailStatistics` in the `Math` section of the command file.

To compute the carrier concentration in [Equation 194](#) the Gaussian band tails, you can apply a specific and accurate approximation developed for the DOS of organic materials (see [Gaussian Density-of-States for Organic Semiconductors on page 322](#)). It can be activated with the `GaussianFromOrganic` option in the band tail statements.

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

To plot the carrier occupation of the band tails  $n_{bt}/n$ , you should use the `eBandTailOccupation` and `hBandTailOccupation` dataset names for electrons and holes, respectively. To plot the carrier occupation of the conduction or valence bands  $n_B/n$ , you should use the `eAllValleyOccupation` and `hAllValleyOccupation` dataset names (see [Using Multivalley Band Structure on page 331](#)).

With an experimental option, you can consider that not all carriers are thermalized to ambient or lattice temperature, for example, due to a forward bias heating. For this option, two carrier populations are accounted for in the carrier concentration computation, where a nonthermalized population will have a user-defined temperature equal to  $T = q\sigma/k$ , where  $\sigma$  is the band tail parameter, and  $k$  is the Boltzmann constant. In this case, the DOS and band structure without band tails are used for both populations, but for the nonthermalized population,  $N_t$  will work as the effective DOS, and  $\epsilon_0$  will work as an additional band-edge shift. To activate this option, specify `TwoPopulation` in the band tail statements.

## Multivalley Band Structure

[Equation 43](#) and [Equation 44](#) give a dependency of electron and hole concentrations on the quasi-Fermi level for a single-valley representation of the semiconductor band structure. Stress-induced change of the silicon band structure and the nature of low bandgap materials require you to consider several valleys in the conduction and valence bands to compute the correct dependency of the carrier concentration on the quasi-Fermi level.

With the parabolic band assumption and Fermi-Dirac distribution function applied to each valley, the multivalley electron and hole concentrations are represented as follows:

$$n = N_C F_{MV,n} \frac{E_{F,n} - E_C}{kT} = N_C \sum_{i=1}^{N_n} g_n^i F_{1/2} \frac{E_{F,n} - E_C - \Delta E_n^i}{kT} \quad (195)$$

$$p = N_V F_{MV,p} \frac{E_V - E_{F,p}}{kT} = N_V \sum_{i=1}^{N_p} g_p^i F_{1/2} \frac{\Delta E_p^i + E_V - E_{F,p}}{kT} \quad (196)$$

where  $N_n$  and  $N_p$  are the electron and hole numbers of valleys,  $g_n^i$  and  $g_p^i$  are the electron and hole DOS valley factors (see [Equation 208](#) and [Equation 209](#)), and  $\Delta E_n^i$  and  $\Delta E_p^i$  are the electron and hole valley energy shifts relative to the band edges  $E_C$  and  $E_V$ , respectively. All other variables in these equations are the same as in [Equation 43](#) and [Equation 44](#). Without Fermi statistics, the Fermi-Dirac integrals  $F_{1/2}$  in these equations are replaced by exponents that correspond to the Boltzmann statistics for each valley.

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

Similarly to the Fermi statistics, the inverse functions of  $F_{MV,n}$  and  $F_{MV,p}$  are used to define the variables  $\gamma_n$  and  $\gamma_p$  that bring an additional drift term with  $\nabla(\ln\gamma)$  as it is expressed in [Equation 47](#) and [Equation 48](#):

$$\gamma_n = \frac{n}{N_C} \exp \left[ -F_{MV,n}^{-1} \frac{n}{N_C} \right] \quad (197)$$

$$\gamma_p = \frac{p}{N_V} \exp \left[ -F_{MV,p}^{-1} \frac{p}{N_V} \right] \quad (198)$$

To compute the inverse functions  $F_{MV,n}^{-1} \frac{n}{N_C}$  and  $F_{MV,p}^{-1} \frac{p}{N_V}$ , an internal Newton solver is used.

Together with the fast Joyce–Dixon approximation of the Fermi–Dirac integral  $F_{1/2}$ , the multivalley model of Sentaurus Device provides an option for a numeric integration over the energy to compute the carrier density using Gauss–Laguerre quadratures. This extends applicability of the model to account for band nonparabolicity (see [Nonparabolic Band Structure](#)) and the MOSFET channel quantization effect using the multivalley MLDA model (see [Modified Local-Density Approximation Model on page 370](#)). To control the accuracy and CPU time of such a numeric integration, it is possible to set a user-defined number of integration points in the Gauss–Laguerre quadratures.

## Nonparabolic Band Structure

The Fermi–Dirac integral  $F_{1/2}$  assumes a typical parabolic band approximation where an energy dependency of the DOS is approximated with  $\sqrt{\varepsilon}$ . Usually, such a simple approximation is valid only in the vicinity of the band minima. Generally, the isotropic nonparabolicity parameter  $\alpha$  is introduced into the dispersion relation as follows:  $\varepsilon(1 + \alpha\varepsilon) = (\hbar k)^2/(2m)$ , where  $k$  is the wavevector, and  $m$  is the effective mass.

With such a dispersion, the multivalley carrier density is computed similarly to [Equation 195](#) and [Equation 196](#), but  $F_{1/2}$  is replaced by the following integral:

$$F_{1/2}^\alpha(\eta, T) = \sum_0^{\infty} \frac{(1 + 2kT\alpha\varepsilon)\sqrt{\varepsilon(1 + kT\alpha\varepsilon)}}{1 + e^{\varepsilon - \eta}} d\varepsilon \quad (199)$$

where  $\eta$  is the carrier quasi-Fermi energy defined by [Equation 49](#) and [Equation 50](#).

Compared to  $F_{1/2}$ , the above integral explicitly depends on the carrier temperature  $T$ , which is accounted for in hydrodynamic and thermodynamic problems.

The more general case of nonparabolic bands can be accounted for with the six-band  $k\cdot p$  band-structure model for holes (see [Strained Hole Effective Mass and DOS on page 948](#)).

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

The model considers three hole bands (the heavy-hole, light-hole, and split-off bands) and the DOS of band  $n$  is given generally by:

$$D_n(\varepsilon) = \frac{2}{(2\pi)^3} \circ \int_{\varepsilon_n(\kappa)=\varepsilon} \frac{dS}{|\nabla_{\kappa}\varepsilon_n(\kappa)|} \quad (200)$$

Therefore, accounting for Fermi–Dirac carrier distribution over the energy, the total hole concentration in the valence band can be expressed as:

$$p(E_{F,p}, T) = \sum_{i=0}^{\infty} \frac{D_i(\varepsilon)}{1 + \exp \left[ \frac{\varepsilon - \Delta E_p^i - E_V + E_{F,p}}{kT} \right]} \quad (201)$$

where variable notations correspond to ones in [Equation 196](#) and integrals over the energy in [Equation 201](#) are computed numerically using Gauss–Laguerre quadrature.

Similar to the parabolic case, without Fermi statistics, the Fermi–Dirac distribution function in [Equation 199](#) and [Equation 201](#) is replaced by the Boltzmann one.

## Bandgap Widening

The bandgap widening effect is related to the carrier geometric confinement, which is important to account for in ultrathin layer semiconductor structures. Typically, the electrons in the  $\Gamma$ -valley of III–V materials have a very small effective mass, which leads to increased geometric confinement in such materials. This effect in the model is accounted for by the simple assumption that the carrier DOS in [Equation 199](#) and [Equation 200](#), and in the Fermi–Dirac integrals of [Equation 195](#) and [Equation 196](#), is zero up to the first subband energy  $\varepsilon_1$  computed analytically for the infinite barrier rectangular quantum well with the layer thickness size  $L_z$ . Mostly, this option is designed to work together with the MLDA quantization model (see [Modified Local-Density Approximation Model on page 370](#)).

Moreover, the multivalley band structure with this bandgap widening model could be used with other quantization models, such as the [Density Gradient Model on page 362](#).

The first subband energy for nonparabolic bands is computed for each valley  $i$  as follows:

$$\varepsilon_1^i = \frac{-1 + \sqrt{1 + \frac{2\alpha^i}{m_q^i} \left( \frac{\hbar\pi}{L_z} \right)^2}}{2\alpha^i} \quad (202)$$

where  $\alpha^i$  is the band nonparabolicity,  $L_z$  is the layer thickness, and  $m_q^i$  is the quantization mass computed automatically along the confinement direction or specified in the valley definition. To account for  $\varepsilon_1^i$  in the density integrals [Equation 199](#) and [Equation 200](#), the DOS energy is replaced by  $\varepsilon + \varepsilon_1^i$ . Various options to compute the layer thickness automatically or to define it explicitly are described in [Geometric Parameters on page 433](#) and [Using the MLDA Model on page 373](#).

## Monte Carlo Density-of-States

To have consistency between the continuity transport models (see [Chapter 8 on page 238](#)) and the Monte Carlo method (see the *Sentaurus™ Device Monte Carlo User Guide*), there is an option to use Monte Carlo DOS in the multivalley model. The Monte Carlo DOS is computed for the full semiconductor band structure used in Monte Carlo simulations.

If the DOS  $D_i(\varepsilon)$  is computed for various bands in the conduction band, the electron density can be expressed as:

$$n(E_{F,n}, T) = \sum_{i=0}^{\infty} \frac{D_i(\varepsilon)}{1 + \exp\left(\frac{\varepsilon + \Delta E_n^i + E_C - E_{F,n}}{kT}\right)} d\varepsilon \quad (203)$$

where  $E_{F,n}$  is the electron quasi-Fermi energy, and  $\Delta E_n^i$  is the difference between the energy of band  $i$  and the conduction band edge. A similar expression works for the hole density (see [Equation 201](#)).

## Non-Three-Dimensional Density-of-States

As known generally for the parabolic bands, 3D DOS  $D_i(\varepsilon)$  in [Equation 201](#) and [Equation 203](#), for example, has  $\varepsilon^{1/2}$  for the energy dependence. However, for thin 2D structures and thin 1D nanowires, a strong geometric confinement modifies the energy dependence in DOS to  $\varepsilon^0$  and  $\varepsilon^{-1/2}$ , respectively. Another case is 2D materials. For graphene-like materials with atomic layers, the energy dependence changes to  $\varepsilon^1$ .

In a general form accounting for nonparabolicity, such a DOS of valley or band  $i$  can be written in the following simple form:

$$D_i(\varepsilon, \alpha, k) = N_k (1 + 2\alpha\varepsilon)^k (1 + \alpha\varepsilon)^k \quad (204)$$

where:

- $D_i(\varepsilon, \alpha, k)$ , with  $k = 1/2, 0, -1/2$ , and  $1$ , represents 3D, 2D, 1D, and 2D material DOS, respectively.
- $N_k$  is a valley or band constant factor, which can be written as follows for all of these DOS cases:

$$\begin{aligned} N_{1/2} &= \frac{\sqrt{2}m_i^{3/2}}{\pi^2\hbar^3} & N_0 &= \frac{m_i}{\pi\hbar^2 L} \\ N_{-1/2} &= \frac{\sqrt{2}m_i}{\pi\hbar L^2} & N_1 &= \frac{2}{\pi\hbar^2 v_F^2 L} \end{aligned} \quad (205)$$

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

where  $m_i$  is the effective DOS mass of valley or band  $i$ ,  $L$  is the layer thickness used in the device simulation, and  $v_F$  is the Fermi velocity.

#### Note:

In [Equation 205](#)  $N_{1/2}$  represents constants in 3D effective DOS, and it is provided solely as a reference.

## Graphene Nanoribbon Density-of-States

Although the graphene nanoribbon (GNR) DOS case comes from [Equation 205](#), it should be considered separately because of its dependence on the band gap. The GNR DOS is written as follows:

$$D_i(\epsilon) = N_1(E_g/2 + \Delta E^i + \epsilon) \quad (206)$$

where  $E_g$  is the band gap,  $\Delta E^i$  is the user-defined energy shift for the  $i$ -band, and  $N_1$  is from [Equation 205](#).

## Carbon Nanotube Density-of-States

The carbon nanotube (CNT) DOS case does not come from [Equation 205](#) and is represented by the CNT DOS in the following form:

$$D_i(\epsilon) = \frac{4}{\pi \hbar v_F A} \left( 1 - \frac{\left[ \frac{E_g/2 + \Delta E^i}{E_g/2 + \Delta E^i + \epsilon} \right]^2 - 1}{2} \right)^{-1/2} \quad (207)$$

where:

- $E_g$  is the band gap.
- $\Delta E^i$  is the user-defined energy shift for the  $i$ -band.
- $v_F$  is the Fermi velocity.
- $A = \pi d L$  is the CNT-conducting area, where  $d$  is the CNT diameter and  $L$  is the thickness used to represent the CNT-conducting area in the device simulation.

---

## Using Multivalley Band Structure

Multivalley statistics is activated with the keyword `Multivalley` in the `Physics` section. If the model must be activated only for electrons or holes, the keywords `eMultiValley` and `hMultiValley` can be used, respectively. The model can be defined regionwise as well.

For testing and comparison purposes, the numeric Gauss–Laguerre integration (mentioned in [Nonparabolic Band Structure on page 328](#)) can be activated for the Fermi–Dirac integral  $F_{1/2}$ .

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

To set such an integration, the keyword `DensityIntegral` must be used as in the following statement:

```
Physics { (e|h)Multivalley(DensityIntegral) }
```

Similarly, to activate the carrier density computation that accounts for nonparabolicity as in [Equation 199](#), the following statement must be used:

```
Physics { (e|h)Multivalley(Nonparabolicity) }
```

To activate the six-band  $k \cdot p$  band-structure model for holes described in [Strained Hole Effective Mass and DOS on page 948](#) and [Nonparabolic Band Structure on page 328](#), you can specify the following:

```
Physics { hMultivalley(kpDOS) }
```

This statement creates three hole bands (light, heavy, and spin-orbit) in the multivalley model. The parameters affecting the six-band  $k \cdot p$  band-structure model (see [Equation 1015](#) for the DOS mass and [Equation 200](#)) include the deformation potentials of the model for hole bands (`adp`, `bdp`, `ddp`, and `dso`) and the Luttinger parameters (`gamma_1`, `gamma_2`, and `gamma_3`). All parameters can be specified in the `LatticeParameters` section of the parameter file and can be mole fraction dependent. These parameters define relaxed hole bands and the stress-induced change (see [Deformation of Band Structure on page 940](#)).

To control the accuracy of surface integration in [Equation 200](#), you can use the following grid setting in  $k$ -space, `Math{ kGridSizeFor6kp(Nphi,Ntheta) }`, where `Nphi` and `Ntheta` define the number of points for an angular spherical coordinate system. Due to symmetry and possibly the stress effect in DOS, half of the  $k$ -space is used, where both angles are changed from 0° to 180°. The default setting is `kGridSizeFor6kp(10,16)`.

To activate the two-band  $k \cdot p$  band-structure model for electrons described in [Strained Electron Effective Mass and DOS on page 945](#) (where three  $\Delta_2$  valleys will be created), specify:

```
Physics { eMultivalley(kpDOS) }
```

The parameters affecting the two-band  $k \cdot p$  band-structure model for  $\Delta_2$  valleys (see [Equation 1002–Equation 1005](#)) include the deformation potentials of the  $k \cdot p$  model for electron bands (`xis`, `dbs`, `xiu`, and `xid`), the Sverdlov  $k \cdot p$  parameter (`Mkp`), and the relaxed nonparabolicity  $\alpha_0$  (this is accounted for only with the `Nonparabolicity` keyword in the `eMultivalley` statement). These parameters can be specified in the `LatticeParameters` section of the parameter file. In addition, the  $\Delta_2$  valley effective masses (`me_10` and `me_t0`) can be defined in the `StressMobility` section. All parameters can be mole fraction dependent. These parameters define three relaxed  $\Delta_2$  valleys and the stress-induced change (see [Deformation of Band Structure on page 940](#)).

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

To control the number of Gauss–Laguerre integration points (the default is 30, which gives a good compromise between accuracy of the numeric integration and CPU time), the global Math statement must be used:

```
Math { DensityIntegral(30) }
```

For non-k·p bands, all multivalley model parameters are defined in the `Multivalley` section of the parameter file.

You can define an arbitrary number of valleys; however, by default, for example for silicon, Sentaurus Device defines three  $\Delta_2$  electron valleys and two hole valleys as follows:

```
Multivalley{
    eValley"Delta1"(1,0,0)(ml=0.914, mt=0.196, energy=0, degeneracy=2,
                           alpha=0.5, xiui=9.16, xid=0.77)
    eValley"Delta2"(0,1,0)(ml=0.914, mt=0.196, energy=0, degeneracy=2,
                           alpha=0.5, xiui=9.16, xid=0.77)
    eValley"Delta3"(0,0,1)(ml=0.914, mt=0.196, energy=0, degeneracy=2,
                           alpha=0.5, xiui=9.16, xid=0.77)
    hValley" LH "(m=0.16, energy=0, degeneracy=1)
    hValley" HH "(m=0.49, energy=0, degeneracy=1)
}
```

The above definition of "Delta" valleys represents a most general way to define valleys in the multivalley band structure. For example, the valley "Delta1" defines an ellipsoidal valley with the effective masses  $m_l = 0.914$  and  $m_{l^*} = 0.196$ , its main axis oriented in the <100> direction, the nonparabolicity  $\alpha = 0.5 \text{ eV}^{-1}$ , the degeneracy equal to 2, zero energy shift  $\Delta E_n$  from the conduction band edge, and the deformation potentials  $\Xi_u, \Xi_d$ , which are used to have a stress effect in the valley energy using the linear deformation potential model (see [Equation 995 in Deformation of Band Structure on page 940](#)).

The general valley definition allows also another parameter  $m_g$  that could redefine the quantization mass computed automatically by the multivalley MLDA model (see [Modified Local-Density Approximation Model on page 370](#)). The hole valleys above are assumed to be simple spherical ones with no stress dependency. This is a huge simplification compared to the six-band k·p model.

#### Note:

With both the `Multivalley(parfile)` and `Multivalley` options, only valleys defined in the `Multivalley` section of the parameter file (or default material-specific ones) will be used. With the `Multivalley` section in the parameter file, all default valleys will be ignored and only valleys defined in the parameter file will be used.

With the `Multivalley(kpDOS)` option in the `Physics` section, all multivalley model parameters ( $N_n, N_p, g_n^i, g_p^i, \Delta E_n^i$ , and  $\Delta E_p^i$ ) are defined and described by stress models in [Multivalley Band Structure on page 950](#). This case ignores all valleys defined in the parameter file.

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

To use the k·p bands together with valleys in the parameter file, specify:

```
Multivalley(kpDOS parfile)
```

The option `eMultivalley(kpDOS parfile)` can be used, for example, for SiGe applications where the two-band k·p model brings three  $\Delta_2$  valleys, but four L-valleys are taken from the parameter file. Generally, it is a useful option to combine k·p and analytic valleys, but you must be careful not to double-count the  $\Delta_2$  valleys because, by default, the parameter file for SiGe has analytic  $\Delta_2$  valleys as well. So, these valleys must be removed from the `Multivalley` section of the parameter file. However, if no `Multivalley` section is defined in the parameter file, the default valleys are still used and there might be double-counting.

To account for the bandgap widening effect as in [Equation 202](#), the `ThinLayer` keyword must be in the `MultiValley` statement, and the `LayerThickness` statement must be set. All `LayerThickness` options are described in [Extracting Layer Thickness on page 379](#) where generally the layer thickness can be extracted automatically for defined regions. In addition, some specific options can be found in [Using the MLDA Model on page 373](#) for non-1D confinement, but practically, if the geometric confinement is mostly 1D, such a setting could be as simple as:

```
LayerThickness( Thickness = LZ)
Multivalley( ThinLayer )
```

To activate the use of the Monte Carlo DOS in the multivalley model, specify one of the following options:

```
Physics { (e|h)Multivalley(mcDOS( "mcdos_file_name" )) }
```

or:

```
Physics { (e|h)Multivalley(mcDOS) }
File { mcDOS="mcdos_file_name" }
```

To save the Monte Carlo DOS file, use `CreateDOSFile=1` or `CreateDOSFile=2` in the `MonteCarlo` statement of the command file (see the *Sentaurus™ Device Monte Carlo User Guide*). The DOS saved in the file can be solely the DOS of the full conduction and valence bands (`CreateDOSFile=1`) or the DOS of separate bands (`CreateDOSFile=2`). For the DOS of separate bands, the multivalley model will search for bands with the same energy minima and will combine such bands into one in the multivalley model (to speed up related numeric density integrations). The saved Monte Carlo DOS file will be named `<MonteCarloOut>.dostot.dat`, where `MonteCarloOut` is an output field of the Monte Carlo simulations in the `File` section of the input file.

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

#### Note:

The `mcDOS` option in the `Multivalley` statement cannot be used with any other option. Therefore, the Monte Carlo DOS (bands) cannot be combined with analytic and  $k \cdot p$  bands, and with the multivalley `MLDA` and `ThinLayer` quantization models.

For any analytic valley, you can use the non-3D DOS option described by [Equation 204](#), [Equation 205](#), [Equation 206](#), and [Equation 207](#). To activate this option, you must set `dospower=k` or `dosmodel=<string>` in the `(e|h)Valley` statement in the parameter file:

- `dospower=0.5` or `dosmodel=3D` is the default and corresponds to 3D DOS.
- `dospower=0` or `dosmodel=2D` specifies 2D DOS.
- `dospower=-0.5` or `dosmodel=1D` specifies 1D DOS.
- `dosmodel=GNR` specifies 2D graphene-like material DOS.
- `dosmodel=CNT` specifies CNT DOS.

All these non-3D `dosmodel` options recompute the constant factor  $N_k$  automatically in the effective DOS with [Equation 205](#). The required layer thickness  $L$  for all of these options (except 3D) must be set by the `LayerThickness` command (see [Extracting Layer Thickness on page 379](#)). The Fermi velocity  $v_F$  needed for the `GNR` and `CNT` options, and the CNT diameter  $d$  for the `CNT` option should be set in the `Bandgap` model parameters (see [Band Gap and Parameters of Non-Three-Dimensional Density-of-States on page 316](#)).

#### Note:

Generally, you can set the parameter `dospower` to any arbitrary number. However, if the value is not equal to specific values in [Equation 205](#), 3D effective DOS will be used.

There are two options for the model parameters  $g_n^i$  and  $g_p^i$ . By default, these parameters are defined by the effective masses from the `(e|h)Valley` statement, and such a definition removes the dependency on  $N_C$  and  $N_V$  (defined by [Effective Masses and Effective Density-of-States on page 319](#)) in [Equation 195](#) and [Equation 196](#):

$$g_{n,p}^i = \frac{2.5094 \times 10^{19}}{N_{C,V}} d^i \sqrt{\frac{m_l^i m_t^i}{m_0 m_0 m_0}} \frac{T_{n,p}}{300 \text{ K}}^{\frac{3}{2}} \quad (208)$$

Another option is to use `Multivalley(RelativeToDOSMass)`. In this case, the parameters  $g_n^i$  and  $g_p^i$  define the carrier density relative to the effective DOS  $N_C$  and  $N_V$ :

$$g_{n,p}^i = \frac{d^i \sqrt{m_l^i m_t^i m_t^i}}{N} \frac{d^k \sqrt{m_l^k m_t^k m_t^k}}{\exp \mp \frac{\Delta E_{n,p}^k}{kT}} \quad (209)$$

$k = 1$

## Chapter 12: Semiconductor Band Structure

### Multivalley Band Structure

where  $\Delta E_n^k$  and  $\Delta E_p^k$  are the valley energy shifts in reference to the band edge (positive for the conduction band and negative for the valence band).

If all valley energy shifts equal zero, [Equation 209](#) provides the single-valley condition where, as usual, only the effective DOS and band edge define the semiconductor band structure and carrier concentration.

The valley definition allows you to have mole fraction-dependent valley parameters. The following example shows the definition for two valleys ( $\Delta_2$  and L) and with two mole fraction intervals in SiGe material:

```
MultiValley{  
    Xmax(0) = 0.0  
    Xmax(1) = 0.85  
    Xmax(2) = 1.0  
  
    eValley"Delta1"(1,0,0)(ml=0.914 mt=0.196 energy=0 alpha=0.5  
                            degeneracy=2 xiui=9.16 xid=-0.77)  
    eValley"L1"(1,1,1)(ml=1.69 mt=0.13 energy=1.1 alpha=0.5 degeneracy=1  
                        xiui=11.5 xid=-6.58)  
    eValley"Delta1"(1)(ml=0.914 mt=0.196 energy=0 alpha=0.5 degeneracy=2  
                      xiui=9.16 xid=-0.77)  
    eValley"L1"(1)(ml=1.768 mt=0.0967 energy=0 alpha=0.5 degeneracy=1  
                      xiui=11.5 xid=-6.58)  
    eValley"Delta1"(2)(ml=0.915 mt=0.201 energy=0.19 alpha=0.5  
                      degeneracy=2 xiui=9.42 xid=-0.54)  
    eValley"L1"(2)(ml=1.768 mt=0.0967 energy=0 alpha=0.5 degeneracy=1  
                      xiui=11.5 xid=-6.58)  
}
```

[Equation 195](#) and [Equation 196](#) give expressions for the total  $n$  and  $i$ -valley  $n_i$  carrier concentrations. The valley occupation function  $n_i/n$  can be plotted using the following keywords in the `Plot` section: `eValleyOccupation_name` and `hValleyOccupation_name`, where `name` should correspond exactly to a valley name defined by the `eValley` or `hValley` statement in the simulation command file. For  $k\cdot p$  bands, you should use `kpDelta1`, `kpDelta2`, and `kpDelta3` as `name` for electrons and `kpHH`, `kpLH`, and `kpSO` as `name` for holes. If an undefined `name` is used, then Sentaurus Device stops the simulation at the plotting stage.

In some cases, using the total occupation of all valleys (the sum of all individual valley occupations) might be useful. It can be plotted by using the `eAllValleyOccupation` and `hAllValleyOccupation` keywords in the `Plot` section. Mostly, it should give a unit value, but when using the band tail model, you can see a carrier redistribution between the conduction (or valence) band and the tail bands (see [Band Tails on page 325](#)). These keywords always work for full conduction and valence bands even when not using the multivalley model.

---

## References

- [1] W. Bludau, A. Onton, and W. Heinke, "Temperature dependence of the band gap in silicon," *Journal of Applied Physics*, vol. 45, no. 4, pp. 1846–1848, 1974.
- [2] H. S. Bennett and C. L. Wilson, "Statistical comparisons of data on band-gap narrowing in heavily doped silicon: Electrical and optical measurements," *Journal of Applied Physics*, vol. 55, no. 10, pp. 3582–3587, 1984.
- [3] J. W. Slotboom and H. C. de Graaff, "Measurements of Bandgap Narrowing in Si Bipolar Transistors," *Solid-State Electronics*, vol. 19, no. 10, pp. 857–862, 1976.
- [4] J. W. Slotboom and H. C. de Graaff, "Bandgap Narrowing in Silicon Bipolar Transistors," *IEEE Transactions on Electron Devices*, vol. ED-24, no. 8, pp. 1123–1125, 1977.
- [5] J. W. Slotboom, "The pn-Product in Silicon," *Solid-State Electronics*, vol. 20, no. 4, pp. 279–283, 1977.
- [6] D. B. M. Klaassen, J. W. Slotboom, and H. C. de Graaff, "Unified Apparent Bandgap Narrowing in n- and p-Type Silicon," *Solid-State Electronics*, vol. 35, no. 2, pp. 125–129, 1992.
- [7] J. del Alamo, S. Swirhun, and R. M. Swanson, "Simultaneous Measurement of Hole Lifetime, Hole Mobility and Bandgap Narrowing in Heavily Doped n-Type Silicon," in *IEDM Technical Digest*, Washington, DC, USA, pp. 290–293, December 1985.
- [8] J. del Alamo, S. Swirhun, and R. M. Swanson, "Measuring and Modeling Minority Carrier Transport in Heavily Doped Silicon," *Solid-State Electronics*, vol. 28, no. 1–2, pp. 47–54, 1985.
- [9] S. E. Swirhun, Y.-H. Kwark, and R. M. Swanson, "Measurement of Electron Lifetime, Electron Mobility and Band-Gap Narrowing in Heavily Doped p-Type Silicon," in *IEDM Technical Digest*, Los Angeles, CA, USA, pp. 24–27, December 1986.
- [10] S. E. Swirhun, J. A. del Alamo, and R. M. Swanson, "Measurement of Hole Mobility in Heavily Doped n-Type Silicon," *IEEE Electron Device Letters*, vol. EDL-7, no. 3, pp. 168–171, 1986.
- [11] J. A. del Alamo and R. M. Swanson, "Measurement of Steady-State Minority-Carrier Transport Parameters in Heavily Doped n-Type Silicon," *IEEE Transactions on Electron Devices*, vol. ED-34, no. 7, pp. 1580–1589, 1987.
- [12] S. C. Jain and D. J. Roulston, "A Simple Expression for Band Gap Narrowing (BGN) in Heavily Doped Si, Ge, GaAs and  $Ge_xSi_{1-x}$  Strained Layers," *Solid-State Electronics*, vol. 34, no. 5, pp. 453–465, 1991.
- [13] U. Lindefelt, "Doping-induced band edge displacements and band gap narrowing in 3C-, 4H-, 6H-SiC, and Si," *Journal of Applied Physics*, vol. 84, no. 5, pp. 2628–2637, 1998.

## Chapter 12: Semiconductor Band Structure

### References

- [14] A. Schenk, "Finite-temperature full random-phase approximation model of band gap narrowing for silicon device simulation," *Journal of Applied Physics*, vol. 84, no. 7, pp. 3684–3695, 1998.
- [15] M. A. Green, "Intrinsic concentration, effective densities of states, and effective mass in silicon," *Journal of Applied Physics*, vol. 67, no. 6, pp. 2944–2954, 1990.
- [16] J. E. Lang, F. L. Madarasz, and P. M. Hemenger, "Temperature dependent density of states effective mass in nonparabolic p-type silicon," *Journal of Applied Physics*, vol. 54, no. 6, p. 3612, 1983.
- [17] G. Paasch and S. Scheinert, "Charge carrier density of organics with Gaussian density of states: Analytical approximation for the Gauss–Fermi integral," *Journal of Applied Physics*, vol. 107, no. 10, p. 104501, 2010.
- [18] E. O. Kane, "Thomas-Fermi Approach to Impure Semiconductor Band Structure," *Physical Review*, vol. 131, no. 1, pp. 79–88, 1963.

# 13

## Incomplete Ionization

---

*This chapter discusses how to account for incomplete ionization in Sentaurus Device.*

---

### Considering Incomplete Ionization

In silicon, with the exception of indium, dopants can be considered to be fully ionized at room temperature because the impurity levels are sufficiently shallow. However, when impurity levels are relatively deep compared to the thermal energy  $kT$ , incomplete ionization must be considered. This is the case for indium acceptors in silicon and nitrogen donors and aluminum acceptors in silicon carbide. In addition, for simulations at reduced temperatures, incomplete ionization must be considered for all dopants. For these situations, Sentaurus Device has an ionization probability model based on activation energy. The ionization (activation) is computed separately for each species present.

All doping species for Sentaurus Device are defined in the `datexcodes.txt` file. To add new doping species or to modify an existing specification, see [Specifying Doping Species on page 60](#).

---

### Using the Incomplete Ionization Model

The incomplete ionization model is activated with the keyword `IncompleteIonization` in the `Physics` section:

```
Physics{ IncompleteIonization }
```

The incomplete ionization model for selected species is activated with the additional keyword `Dopants`:

```
Physics{ IncompleteIonization(Dopants = "Species_name1  
Species_name2 ...") }
```

For example, the following command line activates the model only for boron:

```
Physics{ IncompleteIonization(Dopants = "BoronActiveConcentration") }
```

## Chapter 13: Incomplete Ionization

### Using the Incomplete Ionization Model

The incomplete ionization model can be specified in region or material physics (see [Region-Specific and Material-Specific Models on page 67](#)). In this case, the model is activated only in these regions or materials.

Using `DonorPlusConcentration` and `AccepMinusConcentration` in the `Plot` section of the device, you can visualize the ionized donor and acceptor concentrations, respectively. The net ionized doping concentration can be plotted using `IonizedDopingConcentration` or `DopingConcentration`:

$$\text{IonizedDopingConcentration} = \text{DopingConcentration} - \text{AccepMinusConcentration}$$

Finally, the substitutional (active) net concentration can be plotted using `ActiveDopingConcentration`:

$$\text{ActiveDopingConcentration} = \text{DonorConcentration} - \text{AcceptorConcentration}$$

#### Note:

Incomplete ionization is implemented in terms of traps and, therefore, is affected by setting the trap occupation explicitly (see [Explicit Trap Occupation on page 550](#)). Use named traps to avoid affecting dopant ionization by settings intended for *real* traps only.

---

## Incomplete Ionization Model

The concentration of ionized impurity atoms is given by Fermi–Dirac distribution:

$$N_D = \frac{N_{D,0}}{1 + g_D \exp\left(\frac{E_{F,n} - E_D}{kT}\right)} \quad \text{for } N_{D,0} < N_{D,\text{crit}}$$
 (210)

$$N_A = \frac{N_{A,0}}{1 + g_A \exp\left(\frac{E_A - E_{F,p}}{kT}\right)} \quad \text{for } N_{A,0} < N_{A,\text{crit}}$$
 (211)

where  $N_{D,0}$  and  $N_{A,0}$  are the substitutional (active) donor and acceptor concentrations,  $g_D$  and  $g_A$  are the degeneracy factors for the impurity levels, and  $E_D$  and  $E_A$  are the donor and acceptor ionization (activation) energies.

In the literature [1], incomplete ionization in SiC material has been considered and another general distribution function has been proposed, which can be expressed as:

$$N_D = \frac{N_{D,0}}{1 + G_D(T) \exp\left(\frac{E_{F,n} - E_C}{kT}\right)} \quad (212)$$

## Chapter 13: Incomplete Ionization

Using the Incomplete Ionization Model

$$N_A = \frac{N_{A,0}}{1 + G_A(T) \exp\left(-\frac{E_{F,p} - E_V}{kT}\right)} \quad (213)$$

where  $G_D(T)$  and  $G_A(T)$  are the ionization factors discussed in [2][3]. These factors can be defined by a PMI (see [1] and [Example: Matsuura Incomplete Ionization on page 1475](#)).

By comparing [Equation 210](#) and [Equation 211](#) to [Equation 212](#) and [Equation 213](#), it can be seen that, in the case of the Fermi distribution function, the ionization factors can be written as:

$$G_D(T) = g_D \cdot \exp\left(-\frac{\Delta E_D}{kT}\right), \Delta E_D = E_C - E_D \quad \text{and} \quad (214)$$

$$G_A(T) = g_A \cdot \exp\left(-\frac{\Delta E_A}{kT}\right), \Delta E_A = E_A - E_V$$

In Sentaurus Device, the basic variables are potential, electron concentration, and hole concentration. Therefore, it is more convenient to rewrite [Equation 210](#) and [Equation 211](#) in terms of the carrier concentration instead of the quasi-Fermi levels:

$$N_D = \frac{N_{D,0}}{1 + g_D \frac{n}{n_1}}, \text{ with } n_1 = N_C \exp\left(-\frac{\Delta E_D}{kT}\right) \text{ for } N_{D,0} < N_{D,\text{crit}} \quad (215)$$

$$N_A = \frac{N_{A,0}}{1 + g_A \frac{p}{p_1}}, \text{ with } p_1 = N_V \exp\left(-\frac{\Delta E_A}{kT}\right) \text{ for } N_A < N_{A,\text{crit}} \quad (216)$$

The expressions for  $n_1$  and  $p_1$  in these two equations are valid for Boltzmann statistics and without quantization. If Fermi–Dirac statistics or a quantization model (see [Chapter 14 on page 348](#)) is used, then  $n_1$  and  $p_1$  are multiplied by the coefficients  $\gamma_n$  and  $\gamma_p$  defined in [Equation 47](#) and [Equation 48](#) (see [Fermi Statistics on page 233](#)). For  $N_{D/A} > N_{D/A,\text{crit}}$ , the dopants are assumed to be completely ionized, in which case, every donor and acceptor species is considered in the Poisson equation. The values of  $N_{D,\text{crit}}$  and  $N_{A,\text{crit}}$  can be adjusted in the parameter file of Sentaurus Device.

To provide a complete ionization for high doping concentrations, the donor and acceptor activation energies are effectively reduced by the total doping in the semiconductor [4]. It also suggests to have an additional temperature dependence in the activation energy as a fitting option. This is accounted for in the expressions:

$$\Delta E_D = \Delta E_{D,0} - \alpha_D \cdot N_{\text{tot}}^{1/3} + \beta_D \left[ \frac{300}{T} \right]^{\gamma_D} \frac{kT}{q} \quad (217)$$

$$\Delta E_A = \Delta E_{A,0} - \alpha_A \cdot N_{\text{tot}}^{1/3} + \beta_A \left[ \frac{300}{T} \right]^{\gamma_A} \frac{kT}{q} \quad (218)$$

## Chapter 13: Incomplete Ionization

### Using the Incomplete Ionization Model

where:

- $N_{\text{tot}} = N_{A,0} + N_{D,0}$  is the total doping concentration.
- $T$  is the lattice temperature.
- $\alpha_{A/D}$ ,  $\beta_{A/D}$ , and  $\gamma_{A/D}$  are model fitting parameters.

The development undertaken in [5] and [6] suggests an alternative Altermatt ionization model, which is based on accounting for changes introduced by impurities into the semiconductor density-of-states. This model is still based on [Equation 215](#) and [Equation 216](#), but with some modifications of the degeneracy factors  $g_D$  and  $g_A$ , and the activation energies  $\Delta E_D$  and  $\Delta E_A$ :

$$g'_D = \frac{b_D}{g_D + 1 - b_D \frac{n}{n_1}}, \quad g'_A = \frac{b_A}{g_A + (1 - b_A) \frac{p}{p_1}}, \quad b_{A/D} = \frac{1}{1 + N_{A/D,0}/N_b^{d_b}} \quad (219)$$

$$\Delta E_{A/D} = \frac{\Delta E_{A/D,0}}{1 + (N_{A/D,0}/N_E)^{d_E}} + \beta_{A/D} \left[ \frac{300}{T} - 1 \right] \frac{kT}{q} \quad (220)$$

where  $N_b$ ,  $d_b$ ,  $N_E$ , and  $d_E$  are additional fitting parameters of the model defined for each impurity separately, and the last term in [Equation 220](#) is added similarly to [Equation 217](#) and [Equation 218](#) for fitting purposes.

In transient simulations, the terms:

$$\frac{\partial N_D}{\partial t} = \sigma_D v_{\text{th}}^n \left[ \frac{n_1}{g_D} N_{D,0} - n + \frac{n_1}{g_D} N_D \right] \quad (221)$$

$$\frac{\partial N_A}{\partial t} = \sigma_A v_{\text{th}}^p \left[ \frac{p_1}{g_A} N_{A,0} - p + \frac{p_1}{g_A} N_A \right] \quad (222)$$

are included in the continuity equations, where  $v_{\text{th}}^{n,p}$  denotes the carrier thermal velocities, and  $\sigma_{A/D}$  is the capture or emission cross-section.

For very low temperature applications, a field-enhanced ionization of impurities might play an important role. Generally, the ionization equations, such as [Equation 221](#) and [Equation 222](#), involve an account of the emission and capture carrier fluxes with the conduction band or valence band. These equations are the same as described for the trap model in [Local Capture and Emission Rates on page 555](#). The field-enhanced ionization of the incomplete ionization model enhances the emission cross-section with the following allowed models: [J-Model for Cross Sections on page 556](#), [Hurkx Model for Cross Sections on page 557](#), and [Poole–Frenkel Model for Cross Sections on page 557](#).

## Physical Model Parameters

You can access the values of the dopant level  $E_{A/D, 0}$ , the doping- and temperature-dependent shift parameters  $\alpha_{A/D}$ ,  $\beta_{A/D}$ , and  $\gamma_{A/D}$ , the impurity degeneracy factor  $g_{A/D}$ , and the cross section  $\sigma_{A/D}$  in the parameter set `Ionization`.

For each ionized doping species, the `Ionization` parameter set must contain a separate subsection where the parameters  $E_{A/D, 0}$ ,  $\alpha_{A/D}$ ,  $\beta_{A/D}$ ,  $\gamma_{A/D}$ ,  $g_{A/D}$ , and  $\sigma_{A/D}$  are defined. For example, for the dopant `BoronActiveConcentration`, described in [Specifying Doping Species on page 60](#), for the material silicon, the parameter set is:

```
Material = "Silicon" {
    ...
    Ionization {
        ...
        Species ("BoronActiveConcentration") {
            type = acceptor
            E_0 = 0.045          # [eV]
            alpha = 3.1e-8        # [eV cm]
            beta = 0              # [1]
            gamma = 1             # [1]
            g = 4                # [1]
            Xsec = 1.0e-12        # [cm^2]
            Xsec_formula = 1
            highdop_formula = 1
        }
    }
}
```

You can omit the field `type` because the dopant type is specified in the `datexcodes.txt` file. It is used here for informative purposes only.

To activate the Altermatt ionization model defined by [Equation 219](#) and [Equation 220](#), specify `highdop_formula=2` as well as the additional parameters  $N_b$ ,  $d_b$ ,  $N_E$ , and  $d_E$ . For example, for boron, it should be as follows:

```
Species ("BoronActiveConcentration") {
    highdop_formula = 2
    b_Nref = 6e18          # [cm^-3]
    b_pow = 2.4            # [1]
    E_Nref = 1.7e18         # [cm^-3]
    E_pow = 1.4            # [1]
}
```

## Chapter 13: Incomplete Ionization

### Physical Model Parameters

**Table 42** Default coefficients for incomplete ionization model for dopants in silicon

Symbol	Parameter name	Default value for species *										Unit
		As	P	Sb	B	Al	In	N	NDopant	PDopant		
$E_{A/D, 0}$	E_0	0.054	0.045	0.039	0.045	0.045	0.16	0.045	0.045	0.045	eV	
$\alpha_{A/D}$	alpha	$3.1 \times 10^{-8}$									eVcm	
$\beta_{A/D}$	beta	0									1	
$\gamma_{A/D}$	gamma	1									1	
$g_{A/D}$	g	2	2	2	4	4	4	2	2	4	1	
$\sigma_{A/D}$	Xsec	$1.0 \times 10^{-12}$									cm <sup>2</sup>	
$N_{D,crit}$	NDCrit	$1.0 \times 10^{22}$									cm <sup>-3</sup>	
$N_{A,crit}$	NAcrit	$1.0 \times 10^{22}$									cm <sup>-3</sup>	

Table 43 lists all the silicon default parameters of the Altermatt model in Equation 219 and Equation 220 as suggested by [5] and [6].

**Table 43** Default parameters of Altermatt model

Symbol	Parameter name	Default value for species *			Unit
		As	B	P	
$N_b$	b_Nref	1.4e19	6e18	6e18	cm <sup>-3</sup>
$d_b$	b_pow	3	2.4	2.3	1
$N_E$	E_Nref	4e18	1.7e18	3e18	cm <sup>-3</sup>
$d_E$	E_pow	1.5	1.4	2	1

## Chapter 13: Incomplete Ionization

### Multiple Lattice Sites

#### Note:

Implementation of the Altermatt model in [5] and [6] replaces  $n$ ,  $p$  by the doping concentrations  $N_{D,0}$ ,  $N_{A,0}$ , respectively, in [Equation 219](#) to make the model more stable numerically. It is not recommended in Sentaurus Device, but it is still possible with the `-CarrierDensityInAltermatt` option in the `Math` section.

For experimental and fitting purposes, the activation energies of [Equation 217](#), [Equation 218](#), and [Equation 220](#) can be combined into one for any ionization model defined by `highdop_formula`. It can be activated by using the keyword `CombinedEnergy` in the `Species` section with the following expression:

$$\Delta E_{A/D} = \frac{\Delta E_{A/D,0}}{1 + (N_{A/D,0}/N_E)^{d_E}} - \alpha_{A/D} \cdot N_{\text{tot}}^{1/3} + \beta_{A/D} \left[ \frac{300}{T}^{\gamma_{A/D}} - 1 \right] \frac{kT}{q} \quad (223)$$

To activate the field-enhanced ionization models, use the keyword `xsec_formula`, where:

- `xsec_formula=2` activates the [J-Model for Cross Sections on page 556](#).
- `xsec_formula=3` activates the [Hurkx Model for Cross Sections on page 557](#).
- `xsec_formula=4` activates the [Poole–Frenkel Model for Cross Sections on page 557](#).

See the corresponding sections for details about how to set the parameters of these models.

---

## Multiple Lattice Sites

In certain semiconductors, dopants can occupy different lattice sites, resulting in different activation energies for incomplete ionization. An example is 6H-SiC where nitrogen can occupy either a hexagonal site  $h$ , or a cubic site  $k_1$  or  $k_2$ .

For the most accurate results, the doping concentration at each lattice site must be modeled by a different doping species. This approach works as follows:

- Introduce new doping species in the `datexcodes.txt` file to describe the doping concentrations in each lattice site (see [Specifying Doping Species on page 60](#)).

To model the nitrogen doping in 6H-SiC, you can introduce the following species:

- `Nitrogen_h_Concentration`
- `Nitrogen_k1_Concentration`
- `Nitrogen_k2_Concentration`

- Provide a TDR doping file with the concentrations for the new doping species.
- Provide incomplete ionization model parameters for the new doping species (see [Physical Model Parameters on page 343](#)).

## Chapter 13: Incomplete Ionization

### Multiple Lattice Sites

In many cases, it is sufficient to specify the average occupation probability of the various lattice sites. For example, you might assume that nitrogen doping in 6H-SiC occupies the hexagonal site  $h$  and the cubic sites  $k_1$  or  $k_2$  with equal probability.

Sentaurus Device supports this simplification using a `Split` specification in the command file:

```
Physics {
    IncompleteIonization (
        Split (
            Doping = "NitrogenConcentration"
            Weights = (0.3333 0.3333 0.3334)
        )
    )
}
```

#### Note:

The sum of the weights must be one.

With a `Split` specification, you no longer need to introduce new doping species in the `datexcodes.txt` file. Instead, Sentaurus Device generates the necessary doping species as required and initializes them based on the occupation probabilities. In the example, Sentaurus Device introduces the following doping species automatically:

- `NitrogenConcentration_split1`
- `NitrogenActiveConcentration_split1`
- `NitrogenPlusConcentration_split1`
- `NitrogenConcentration_split2`
- `NitrogenActiveConcentration_split2`
- `NitrogenPlusConcentration_split2`
- `NitrogenConcentration_split3`
- `NitrogenActiveConcentration_split3`
- `NitrogenPlusConcentration_split3`

#### Note:

The generated doping species also can be used in a device plot (see [Device Plots on page 177](#)).

## Chapter 13: Incomplete Ionization

### References

The corresponding parameters can then be specified in the parameter file as described in [Physical Model Parameters on page 343](#):

```
Ionization {
    Species ("NitrogenConcentration_split1") {
        E_0 = 0.1
        alpha = 2.5e-8
        g = 2
        Xsec = 1e-12
    }
    ...
}
```

---

## References

- [1] H. Matsuura, "Influence of Excited States of Deep Acceptors on Hole Concentration in SiC," in *International Conference on Silicon Carbide and Related Materials (ICSCRM)*, Tsukuba, Japan, pp. 679–682, October 2001.
- [2] P. Y. Yu and M. Cardona, *Fundamentals of Semiconductors: Physics and Materials Properties*, Berlin: Springer, 2nd ed., 1999.
- [3] K. F. Brennan, *The Physics of Semiconductors: With applications to optoelectronic devices*, Cambridge: Cambridge University Press, 1999.
- [4] M. R. Shaheed and C. M. Maziar, "A Physically Based Model for Carrier Freeze-out in Si- and SiGe- Base Bipolar Transistors Suitable for Implementation in Device Simulators," in *Proceedings of the 1994 Bipolar/BiCMOS Circuits and Technology Meeting (BCTM)*, Minneapolis, MN, USA, pp. 191–194, October 1994.
- [5] P. P. Altermatt, A. Schenk, and G. Heiser, "A simulation model for the density of states and for incomplete ionization in crystalline silicon. I. Establishing the model in Si:P," *Journal of Applied Physics*, vol. 100, no. 11, p. 113714, 2006.
- [6] P. P. Altermatt *et al.*, "A simulation model for the density of states and for incomplete ionization in crystalline silicon. II. Investigation of Si:As and Si:B and usage in device simulation," *Journal of Applied Physics*, vol. 100, no. 11, p. 113715, 2006.

# 14

## Quantization

---

*This chapter describes the features that Sentaurus Device offers to model quantization effects.*

Some features of current MOSFETs (oxide thickness and channel width) have reached quantum-mechanical length scales. Therefore, the wave nature of electrons and holes can no longer be neglected. The most basic quantization effects in MOSFETs are the shift of the threshold voltage and the reduction of the gate capacity. Tunneling, another important quantum effect, is described in [Chapter 24 on page 819](#).

---

### Modeling Quantization Effects

To include quantization effects in a classical device simulation, Sentaurus Device introduces a potential-like quantity  $\Lambda_n$  in the classical density formula:

$$n = N_C F_{1/2} \frac{E_{F,n} - E_C - \Lambda_n}{kT_n} \quad (224)$$

An analogous quantity  $\Lambda_p$  is introduced for holes.

The most important effects related to the density modification (due to quantization) can be captured by proper models for  $\Lambda_n$  and  $\Lambda_p$ . Other effects (for example, single electron effects) exceed the scope of this approach.

Sentaurus Device implements different quantization models for  $\Lambda_n$  and  $\Lambda_p$ , which differ in physical sophistication, numeric expense, and robustness:

- The van Dort model is a numerically robust, fast, and proven model (see [The van Dort Model on page 349](#)). It is suited only to bulk MOSFET simulations. While important terminal characteristics are well described by this model, it does not give the correct density distribution in the channel.
- The 1D Schrödinger equation is a physically sophisticated quantization model (see [1D Schrödinger Equation on page 351](#)). It can be used for MOSFET simulation, and quantum well and ultrathin SOI simulation. Simulations with this model tend to be slow and often lead to convergence problems, which restrict its use to situations with small

## Chapter 14: Quantization

### The van Dort Model

current flow. Therefore, the Schrödinger equation is used mainly for the validation and calibration of other quantization models.

- For 3D structures, an external 2D Schrödinger solver can be used to provide a quantization model (see [External 2D Schrödinger Solver on page 357](#)). This is the physically most sophisticated approach, but requires a high computation time and a complicated setup, and provides reduced numeric robustness.
- For 3D structures, an external Boltzmann transport equation solver can be used to provide a quantum-correction potential as well as an effective mobility to be used in a subdomain of the Sentaurus Device mesh (see [External Boltzmann Solver on page 359](#)). Similar to an external 2D Schrödinger solver, it is a physically sophisticated and computationally demanding approach with reduced numeric robustness.
- The density gradient model (see [Density Gradient Model on page 362](#)) is numerically robust, but significantly slower than the van Dort model. It can be applied to MOSFETs, quantum wells and SOI structures, and gives a reasonable description of terminal characteristics and charge distribution inside a device. Compared to the other quantization models, it can describe 2D and 3D quantization effects.
- The modified local-density approximation (MLDA) model (see [Modified Local-Density Approximation Model on page 370](#)) is a numerically robust and fast model. It can be used for bulk MOSFET simulations and thin SOI simulations. Although it sometimes fails to calculate the accurate carrier distribution in the saturation regions because of its one-dimensional characteristic, it is suitable for 3D device simulations because of its numeric efficiency.
- The quantum-well quantization model (see [Quantum-Well Quantization Model on page 378](#)) can be applied to semiconductor quantum wells with a width that does not vary very much over the device. It approximates the solution by the solution for a trapezoidal potential.

---

## The van Dort Model

The van Dort model [1] computes  $\Lambda_n$  of [Equation 224](#) as a function of  $|\hat{n} \cdot \vec{F}|$ , the electric field normal to the semiconductor–insulator interface:

$$\Lambda_n = \frac{13}{9} \cdot k_{\text{fit}} \cdot G(r) \cdot \left[ \frac{\epsilon \epsilon_0}{4kT} \right]^{1/3} \cdot \left| \hat{n} \cdot \vec{F} - E_{\text{crit}} \right|^{2/3} \quad (225)$$

and likewise for  $\Lambda_p$ . Here,  $k_{\text{fit}}$  and  $E_{\text{crit}}$  are fitting parameters.

## Chapter 14: Quantization

### The van Dort Model

The function  $G(\vec{r})$  is defined by:

$$G(\vec{r}) = \frac{2 \cdot \exp -a^2(\vec{r})}{1 + \exp -2a^2(\vec{r})} \quad (226)$$

where  $a(\vec{r}) = l(\vec{r})/\lambda_{\text{ref}}$  and  $l(\vec{r})$  is a distance from the point  $\vec{r}$  to the interface. The parameter  $\lambda_{\text{ref}}$  determines the distance to the interface up to which the quantum correction is relevant. The quantum correction is applied to those carriers (electrons or holes) that are drawn towards the interface by the electric field; for the carriers driven away from the interface, the correction is 0.

---

## Using the van Dort Model

To activate the model for electrons or holes, specify `eQCvanDort` or `hQCvanDort` in the `Physics` section. For example:

```
Physics { ... eQCvanDort }
```

In [Table 44](#), the default value of  $\lambda_{\text{ref}}$  is from the literature [1]. Other parameters were obtained by fitting the model to experimental data for electrons [1] and holes [2].

*Table 44 Default parameters for van Dort quantum correction model*

Symbol	Electrons	Holes	Unit
$k_{\text{fit}}$	<code>eFit</code>	$2.4 \times 10^{-8}$	<code>hFit</code>
$E_{\text{crit}}$	<code>eEcritQC</code>	$10^5$	<code>eEcritQC</code>
$\lambda_{\text{ref}}$	<code>dRef</code>	$2.5 \times 10^{-6}$	<code>dRef</code>

By default, in [Equation 225](#),  $\hat{n}$  is the normal to the closest semiconductor–insulator interface. The interface can be specified explicitly by `EnormalInterface` in the `Math` section (see [Normal to the Interface on page 426](#)).

---

## 1D Schrödinger Equation

To use the 1D Schrödinger equation:

1. Construct a special-purpose *nonlocal* mesh (see [Nonlocal Mesh for the 1D Schrödinger Equation on page 351](#)).
2. Activate the Schrödinger solver on the nonlocal line mesh with appropriate parameters (see [Using the 1D Schrödinger Equation on page 352](#)).

Sometimes, especially for heteromaterials, the physical model parameters must be adapted (see [Parameters of the 1D Schrödinger Equation on page 352](#)).

**Note:**

The 1D Schrödinger equation is time consuming and often causes converge problems. Furthermore, small-signal analysis (see [Small-Signal AC Analysis on page 149](#)) and noise and fluctuation analysis (see [Chapter 23 on page 781](#)) are not possible when using this solver.

---

## Nonlocal Mesh for the 1D Schrödinger Equation

The specification of the nonlocal mesh determines where in the device the 1D Schrödinger equation can be solved. This section summarizes the features that are typically needed to obtain a correct nonlocal mesh for the 1D Schrödinger equation. For more information about constructing nonlocal meshes, see [Nonlocal Meshes on page 202](#).

To generate a nonlocal mesh, specify `NonLocal` in the global `Math` section. Options to `NonLocal` control the construction of the nonlocal mesh. The following example generates a nonlocal mesh named "NLM" at the interface between regions `gateoxide` and `channel`:

```
Math {
    NonLocal "NLM" (
        RegionInterface="gateoxide/channel"
        Length=10e-7
        Permeation=1e-7
        Direction=(0 1 0) MaxAngle=5
        -Transparent(Region="gateoxide")
    )
}
```

The nonlocal lines extend 10 nm (according to `Length`) to one side of the interface and 1 nm (according to `Permeation`) to the other side. The segments of the nonlocal lines on the two sides are handled differently; see below. In the example, `Direction` and `MaxAngle` restrict the nonlocal mesh to lines that run along the y-axis, with a tolerance of 5°.

`Direction` defines a vector that is typically perpendicular to the interface; therefore, the specification above is appropriate for an interface in the xz plane.

## Chapter 14: Quantization

### 1D Schrödinger Equation

For nonlocal meshes constructed for the Schrödinger equation, the `-Transparent` switch is important. In this example, it suppresses the construction of nonlocal lines for which the section with length `Length` passes through `gateoxide`. Therefore, Sentaurus Device only constructs nonlocal lines that extend 10 nm into `channel` and 1 nm into `gateoxide`, and suppresses the nonlocal lines that extend 1 nm into `channel` and 10 nm into `gateoxide`. Without the `-Transparent` switch, Sentaurus Device would construct both line types and, therefore, would solve the Schrödinger equation not only in the channel, but also in the poly gate (provided that a poly gate exists and `gateoxide` is thinner than 10 nm).

---

## Using the 1D Schrödinger Equation

To activate the 1D Schrödinger equation for electrons or holes, specify `Electron` or `Hole` as an option to `Schroedinger` in the global `Physics` section.

The following example activates the Schrödinger equation for electrons on the nonlocal mesh named "NLM":

```
Physics {  
    Schroedinger "NLM" (Electron)  
}
```

Additional options to `Schroedinger` determine numeric accuracy, details of the density computation, and which eigenstates will be computed. [Table 326 on page 1697](#) lists the optional keywords.

`MaxSolutions`, `Error`, and `EnergyInterval` support the option `electron` or `hole`. For example, the specification `MaxSolutions=30` sets the value for both electrons and holes. The specifications `MaxSolutions(electron)=2` and `MaxSolutions(hole)=3` set different values for electrons and holes.

By default, Sentaurus Device only computes bound states, that is, states with an energy that is below the largest value of the potential both left and right of the point with minimal potential. If, according to this definition, no bound states exist, the quantum corrections will be disabled. With the `EnergyInterval` parameter, this behavior can be changed to compute all states with an energy up to the specified value above the lowest potential point.

For backward compatibility, you can define `Schroedinger` in an interface-specific or a contact-specific `Physics` section; in this case, omit the nonlocal mesh name. The specification applies to an unnamed nonlocal mesh that has been constructed for the same location (see [Unnamed Meshes on page 208](#)).

---

## Parameters of the 1D Schrödinger Equation

Typically, the Schrödinger equation must be solved repeatedly, with different masses and potential profiles, resulting in a number of distinct *ladders* of eigenenergies. For example,

## Chapter 14: Quantization

### 1D Schrödinger Equation

the six-fold degenerate, anisotropic conduction band valleys in silicon require the Schrödinger equation to be solved up to three times with different parameters.

Sentaurus Device allows you to specify the number of ladders and the associated parameters explicitly, or it can extract the number of ladders and the parameters from other parameters and the nonlocal line direction automatically.

Parameters for the Schrödinger equation are region specific. The Schrödinger equation must not be solved across regions with fundamentally different band structures. Sentaurus Device displays an error message if band structure compatibility is violated.

## Explicitly Specifying Ladders

Any number of `eLadder`( $m_{z,v}$ ,  $m_{xy,v}$ ,  $d_v$ ,  $\Delta E_v$ ) specifications (for electrons) or `hLadder`( $m_{z,v}$ ,  $m_{xy,v}$ ,  $d_v$ ,  $\Delta E_v$ ) specifications (for holes) is possible in the parameter set `SchroedingerParameters`. The  $v$ -th specification for a particular carrier type defines the quantization mass  $m_{z,v}$ , the mass perpendicular to the quantization direction  $m_{xy,v}$ , the ladder degeneracy  $d_v$ , and a nonnegative band edge shift  $\Delta E_v$ .

The parameter pair `ShiftTemperature` in the `SchroedingerParameters` parameter set is given in kelvin and determines the temperatures  $T_{\text{ref}}$  for electrons and holes that enter scaling factors applied to  $m_{xy,v}$ . The scaling factors ensure that the masses for the Schrödinger equation are consistent with the density-of-states masses. For electrons, the scaling factor is:

$$\frac{m_n^{3/2}}{d_v m_{xy,v} m_{z,v}^{1/2} \exp(-\Delta E_v / kT_{\text{ref}})} \quad (227)$$

and likewise for holes.  $T_{\text{ref}}$  defaults to a large value. When  $T_{\text{ref}} = 0$ , scaling is disabled.

## Automatically Extracting Ladder Parameters

When no explicit ladder specification is present, Sentaurus Device attempts to extract the required parameters automatically. The `formula` parameter pair in the parameter set `SchroedingerParameters` specifies which expressions for the masses to use. If an explicit ladder specification is present, then `formula` for the respective carrier type is ignored, and only the explicit ladder specification is accounted for.

For each carrier, if the `formula` is 0, Sentaurus Device uses the isotropic density-of-states mass as both the quantization mass and the mass perpendicular to it.

For electrons, if the `formula` is 1, Sentaurus Device uses the anisotropic (silicon-like) masses specified in the `eDOSMass` parameter set.

## Chapter 14: Quantization

### 1D Schrödinger Equation

The quantization mass for the conduction band valley  $v$  reads:

$$\frac{1}{m_{z,v}} = \sum_{i=1}^3 \frac{z_i^2}{m_{i,v}} \quad (228)$$

where  $m_{i,v}$  are the effective mass components for valley  $v$ , and  $z_i$  are the coefficients of the unit normal vector pointing in the quantization direction, expressed in the crystal coordinate system.

The `LatticeParameters` parameter set determines the relation to the mesh coordinate system (see [Crystal and Simulation Coordinate Systems on page 886](#)).

For holes, ‘warped’ band structure (formula 1), or heavy-hole and light-hole masses (formula 2) are available (see [Table 45](#)).

*Table 45 Parameters for hole effective masses in Schrödinger solver*

Formula	Symbol	Parameter name	Default value	Unit
1	A	A	4.22	1
	B	B	0.6084	1
	C	C	23.058	1
2	$m_l$	ml	0	1
	$m_h$	mh	0	1

For the former, the quantization mass is given by:

$$m_{z,v} = \frac{m_0}{A \pm \sqrt{B + C \cdot z_1^2 z_2^2 + z_2^2 z_3^2 + z_3^2 z_1^2}} \quad (229)$$

For the latter,  $ml$  and  $mh$  (given as multiples of  $m_0$ ) determine directly the masses for the light-hole and heavy-hole bands.

The masses  $m_{xy,v}$  are chosen to achieve the same density-of-states mass as used in classical simulations. For electrons:

$$m_{xy,v} = \frac{m_n^{3/2}}{m_{z,v}^{1/2} n_v} \quad (230)$$

where  $n_v$  is the number of band valleys (or bands). The expression for holes is analogous.

## Chapter 14: Quantization

### 1D Schrödinger Equation

SchroedingerParameters offers a parameter pair `offset` to model the lifting of the degeneracy of band minima in strained materials. Unless exactly two ladders exist, `offset` must be zero. Positive offsets apply to the electron ladder with lower degeneracy or the heavy-hole band; negative values apply to the electron ladder of higher degeneracy or the light-hole band. The modulus of the value is added to the band edge for the respective ladder, moving it away from mid-gap, while the other ladder remains unaffected. For holes, the shift is taken into account in the scaling of  $m_{xy, v}$  as it is in [Equation 227](#).

---

## Visualizing the Solutions

To visualize the results obtained by the 1D Schrödinger equation, Sentaurus Device offers special keywords for the `NonLocalPlot` section (see [Visualizing Data Defined on Nonlocal Meshes on page 204](#)).

To plot the wavefunctions, specify `WaveFunction` in the `NonLocalPlot` section. Sentaurus Device plots wavefunctions in units of  $\mu\text{m}^{-1/2}$ . To plot the eigenenergies, specify `EigenEnergy`; Sentaurus Device plots them in units of eV. The names of the curves in the output have two numeric indices. The first index denotes the ladder index and the second index denotes the number of zeros of the wavefunction (see `v` and `j` in [Equation 231](#)).

Without further specification, Sentaurus Device will plot all eigenenergies and wavefunctions it has computed. The `Electron` and `Hole` options to `WaveFunction` and `EigenEnergy` restrict the output to the wavefunctions and eigenenergies for electrons and holes, respectively.

The `Electron` and `Hole` options have a sub-option `Number=<int>` to restrict the output to a certain number of states of lowest energy.

The following example plots all hole wavefunctions and the three lowest-energy electron wavefunctions for the nonlocal mesh line close to the coordinate (0, 0, 0):

```
NonLocalPlot(
    (0 0)
) {
    WaveFunction(Electron(Number=3) Hole)
}
```

To plot the overlap integrals between all pairs of the computed wavefunctions on a nonlocal line, specify `OverlapIntegral` in the `NonLocalPlot` section.

---

## 1D Schrödinger Equation

Omitting x- and y-coordinates for simplicity, the 1D Schrödinger equation for electrons is:

$$-\frac{\partial}{\partial z} \frac{\hbar^2}{2m_{z,v}(z)} \frac{\partial}{\partial z} + E_C(z) + \Delta E_v \Psi_{j,v}(z) = E_{j,v} \Psi_{j,v}(z) \quad (231)$$

## Chapter 14: Quantization

### 1D Schrödinger Equation

where:

- $z$  is the quantization direction (typically, the direction perpendicular to the silicon–oxide interface in a MOSFET).
- $v$  labels the ladder.
- $\Delta E_v$  is the (region-dependent) energy offset for the ladder  $v$ .
- $m_{z,v}$  is the effective mass component in the quantization direction.
- $\Psi_{j,v}$  is the  $j$ -th normalized eigenfunction.
- $E_{j,v}$  is the  $j$ -th eigenenergy.

From the solution of this equation, the density is computed as:

$$n(z) = \frac{kT(z)}{\pi\hbar^2} \sum_{j,v} |\Psi_{j,v}(z)|^2 d_v m_{xy,v}(z) \left[ F_0 \left[ \frac{E_{F,n}(z) - E_{j,v}}{kT(z)} \right] - F_0 \left[ \frac{E_{F,n}(z) - E_{\max,v}}{kT(z)} \right] \right] + \sum_v n_{cl,v} \quad (232)$$

where  $m_{xy,v}$  is the mass component perpendicular to the quantization direction and  $d_v$  is the degeneracy for ladder  $v$ . For the parameters  $m_{z,v}$ ,  $m_{xy,v}$ ,  $d_v$ , and  $\Delta E_v$ , see [Parameters of the 1D Schrödinger Equation on page 352](#).

If `DensityTail=MaxEnergy`, then  $E_{\max,v}$  in [Equation 232](#) is the energy of the highest computed subband for ladder  $v$ ; otherwise, it is an estimate of the energy of the lowest not computed subband.  $n_{cl,v}$  is the contribution to the classical density for ladder  $v$  by energies above  $E_{\max,v}$ .

At the ends of the nonlocal line, Sentaurus Device optionally smoothly blends from the classical density to the density according to [Equation 232](#). Equating the resulting density to [Equation 224](#) determines  $\Lambda_n$ .

The Schrödinger equation is solved over a finite domain  $[z_-, z_+]$ . At the endpoints of this domain, the boundary condition:

$$\frac{\Psi'_{j,v}}{\Psi_{j,v}} = \mp \frac{\sqrt{2m_{z,v}|E_{j,v} - E_C|}}{\hbar} \quad (233)$$

is applied, where for the upper sign, all position-dependent functions are taken at  $z_+$  and, for the lower sign, at  $z_-$ .

---

## Notes on the Use of the 1D Schrödinger Equation

Note that:

- Convergence problems: Near the flatband condition, minor changes in the band structure can change the number of bound states between zero and one. By default, Sentaurus Device computes only bound states and, therefore, this change switches from a purely classical solution to a solution with quantization. To avoid the large change in density that this transition can cause, set `EnergyInterval` to a nonzero value (for example, 1). Then, a few unbound states are computed and a hard transition is avoided.
- The Schrödinger solver uses an iterative method. If this iterative method fails to converge, the solution of the coupled system of all equations will not be accepted, even if the RHS and the error as reported in the log file are small.
- Always include a part of the ‘barrier’ in the nonlocal line on which the Schrödinger equation is solved. In a MOSFET, besides the channel region, solve the Schrödinger equation in a part of the oxide adjacent to the channel (for example, 1 nm deep). Use the `Permeation` option to `NonLocal` to achieve this (see [Nonlocal Mesh for the 1D Schrödinger Equation on page 351](#)). Failure to solve in a part of the oxide adjacent to the channel blinds the Schrödinger solver to the barrier. It does not know the electrons are confined and the quantization effects are not obtained.

---

## External 2D Schrödinger Solver

For 3D structures, Sentaurus Device can connect to external 2D Schrödinger solvers to compute the quantum-mechanical density correction. Sentaurus Device interpolates solution-dependent data, such as band edges, quasi-Fermi potentials, and temperature, to 2D cross sections of the device, and passes it to the 2D Schrödinger solvers. From this information and their own configuration, the 2D Schrödinger solvers compute quantum-mechanical densities and pass them back to Sentaurus Device. Using these densities and [Equation 224](#), Sentaurus Device computes the quantum potentials  $\Lambda_n$  and  $\Lambda_p$ . Then, the quantum potentials are interpolated to the 3D device mesh and are used in subsequent calculations.

The connection to the external 2D Schrödinger solver is defined in the global `Physics` section:

```
ExternalSchroedinger <string> (
    NumberOfSlices=<int>           * required parameter, at least two
    Carriers = <carrierlist>        * required parameter
    Volume=<string>                * optional
    SBandCommandFile=<string>       * optional
    DampingLength=<float>           * in um, default 0.005
    MaxMismatch=<float>             * in um, default 1e-4
)
```

The string after `ExternalSchroedinger` is a name used to establish the connection to the 2D Schrödinger solver processes. Two Sentaurus Device processes running concurrently on the same computer, in the same working directory, must not use the same value for this name.

`NumberOfSlices` is the number of 2D slices (cross sections) of the 3D mesh on which to solve the 2D Schrödinger equation. The slices themselves are provided to Sentaurus Device by the 2D Schrödinger solver processes. They must correspond to cuts of the 3D mesh Sentaurus Device is using, that is, at the same point, the 3D mesh and the 2D slices must have the same region. Sentaurus Device checks this condition, with a tolerance that can be changed with the `MaxMismatch` parameter.

To allow users to perform 3D simulations of a symmetric structure with a reduced mesh, the interface to the external 2D Schrödinger solver takes the `ThinLayer(Mirror)` specification into account (see [Geometric Parameters of LayerThickness Command on page 381](#)). Note, however, that the 2D Schrödinger solver still needs the full 2D slices.

Slices are numbered from zero to `NumberOfSlices`-1. This number is used to refer to slices individually, for example, in error messages. The number also is used by the 2D Schrödinger solvers to identify a particular slice.

The carrier types to which you apply a quantum correction are specified by `Carriers`. Possible values are `(Electron)`, `(Hole)`, and `(Electron Hole)` to apply the correction to electrons only, holes only, or both, respectively.

The quantum correction is only applied to semiconductors. If specified, the application is further restricted to the named volume given by `volume` (for named volumes, see [Random Band Edge Fluctuations on page 794](#)). Within that domain, the quantum correction from the 2D slices is interpolated to the volume enclosed by the slices. With increasing distance from the enclosed volume, the quantum correction is damped. For vertices farther outside the enclosed volume than the distance specified with `DampingLength`, the 2D Schrödinger quantum corrections  $\Lambda_n$  and  $\Lambda_p$  are zero.

**Note:**

For typical MOSFET applications, the 2D Schrödinger solver is used mostly to account for the quantization in the active channel area. However, source and drain regions can use other simpler quantum-correction models such as density gradient (see [Density Gradient Model on page 362](#)) or MLDA (see [Modified Local-Density Approximation Model on page 370](#)). For such a combination, the `DampingLength` parameter is also used in the density gradient or MLDA quantum corrections ( $\Lambda_{nO}$  and  $\Lambda_{pO}$ ) to have a smooth transition between these corrections. The parameter defines a position-dependent damping  $\alpha_{SE}$  (which is equal to 1 inside the 2D Schrödinger domain and goes to 0 at `DampingLength` distance from that domain), and the combined quantum corrections are computed as  $\alpha_{SE}\Lambda_n + (1 - \alpha_{SE})\Lambda_{nO}$  and  $\alpha_{SE}\Lambda_p + (1 - \alpha_{SE})\Lambda_{pO}$ .

## Chapter 14: Quantization

### External Boltzmann Solver

If `SBandCommandFile` is used to provide the name of a command file for Sentaurus Band Structure, then Sentaurus Device automatically starts Sentaurus Band Structure processes with this command file. Sentaurus Device passes to each process the number of the slice it handles on the command line.

Otherwise, users are responsible for starting the 2D Schrödinger solver processes, which must be started in the same working directory on the same computer as the Sentaurus Device process that connects to them.

See *Sentaurus™ Device Monte Carlo User Guide*, Using Sentaurus Band Structure as an External Schrödinger Solver for Sentaurus Device, for details about the 2D Schrödinger solver.

---

## Notes on the Use of External 2D Schrödinger Solvers

One crucial step in using the interface to external 2D Schrödinger solvers is to decide the number and the placement of the slices. Usually, the quantum corrections are applied to the channel of a device. The slices at least must resolve the shape of the cross section of the channel.

Using the quantum potentials  $\Lambda_n$  and  $\Lambda_p$  as the quantities that are interpolated between slices, interpolation becomes insensitive to potential and quasi-Fermi potential drops along the channel. Therefore, for devices with a uniform channel cross section, one slice close to either end of the channel should be sufficient to obtain sufficient accuracy.

#### Note:

The placement of the slices is not specified in Sentaurus Device. The only information specified in the Sentaurus Device command file about the slices is their number. The placement of the slices is the responsibility of the external 2D Schrödinger solvers. Therefore, their configuration must be consistent with the 3D mesh used by Sentaurus Device.

Similarly, the exact model for quantization, as well as band structure information beyond the band edges, is not specified by Sentaurus Device, but by the configuration of the 2D Schrödinger solvers.

---

## External Boltzmann Solver

For 3D structures, Sentaurus Device can connect to Sentaurus Device QTX using the Subband-BTE solver to obtain the extracted quantum-correction potential. For coupled carrier–Poisson simulations, Sentaurus Device can also use an effective mobility extracted from the Subband-BTE solver. Similar to the approach described in [External 2D Schrödinger Solver on page 357](#), Sentaurus Device interpolates solution-dependent data, such as band edges, quasi-Fermi potentials, and temperature, to 2D cross sections of the device, and

## Chapter 14: Quantization

### External Boltzmann Solver

passes it to the Subband-BTE solver. From this information and its own configuration, the Subband-BTE solver computes the quantum-correction potential and the effective mobility and passes them back to Sentaurus Device for solving the coupled carrier–Poisson equations. Solution-dependent data is exchanged only on 2D cross sections, and the respective simulator interpolates it to the 3D device mesh as needed.

The connection to the external Boltzmann solver is defined in the global `Physics` section:

```
ExternalBoltzmannSolver <string> (
    NumberOfSlices=<int>                      * required parameter, at least two
    Carriers = ( Electron | Hole )                * required parameter
    Mobility ( Electron | Hole )                  * optional
    ContactName=<string>                        * optional
    SBTECommandFile=<string>                     * optional
)
```

The string after `ExternalBoltzmannSolver` is a name used to establish the connection to the Subband-BTE solver process. Two Sentaurus Device processes running concurrently on the same computer, in the same working directory, must not use the same value for this name.

`NumberOfSlices` is the number of 2D slices (cross sections) of the 3D mesh on which data is exchanged between the Sentaurus Device and Subband-BTE solver processes. The slices themselves are provided to Sentaurus Device by the Subband-BTE solver. They must correspond to cuts of the 3D mesh Sentaurus Device is using, that is, at the same point, the 3D mesh and the 2D slices must have the same region.

#### Note:

Contrary to simulations using the external 2D Schrödinger solver, all slices for a given connection are handled by a single Subband-BTE solver process.

The keyword `Carriers` specifies the carrier type, either electron or holes, for which the Boltzmann transport equation is solved.

Unless `Mobility` is specified, only the quantum-correction potential computed by the Subband-BTE solver is passed to Sentaurus Device. Specifying `Mobility (<carrier>)` also triggers the Subband-BTE solver to extract the effective carrier mobility and to pass it to Sentaurus Device where it is applied to the 3D domain enclosed by the slices. Within that domain, the application of any other mobility model specified in the Sentaurus Device command file is suppressed.

Alternatively, if you specify the following statement, the effective mobility extracted by the Subband-BTE solver can complement certain mobility models specified in the Sentaurus Device command file:

```
Mobility (<carrier>(-DisableInternalMobilityModels))
```

## Chapter 14: Quantization

### External Boltzmann Solver

Supported mobility models are:

- Carrier-carrier scattering
- Mobility degradation at interfaces
- Thin-layer mobility
- High-field saturation
- Ballistic mobility

Constant and doping-dependent mobility degradation models in Sentaurus Device cannot be combined with the effective mobility provided by the Subband-BTE solver.

To allow the Subband-BTE solver to output current–voltage characteristics, you can select a specific `ContactName` in the Sentaurus Device command file for which Sentaurus Device sends the value of the current to the Subband-BTE solver.

Additional optional keywords such as `Volume`, `MaxMismatch`, and `DampingLength` allow you to further control the interpolation of the quantum-correction potential and the effective mobility to the 3D mesh of Sentaurus Device and are described in [External 2D Schrödinger Solver on page 357](#).

**Note:**

`DampingLength` is interpreted differently for the interpolation of the quantum-correction potential and the effective mobility. For the former, the same behavior as described in [External 2D Schrödinger Solver](#) applies; whereas, for the latter, the damping length specifies the distance from the enclosed domain at which there is a discrete transition from the effective mobility extracted by the Subband-BTE solver and the mobility according to the mobility model specification in the Sentaurus Device command file.

If `SBTECommandFile` is used to provide the name of a command file for the Subband-BTE solver, Sentaurus Device automatically starts the solver process with this command file.

Otherwise, users are responsible for starting the Subband-BTE solver process, which must be started in the same working directory on the same computer as the Sentaurus Device process that connects to it.

See [Sentaurus™ Device QTX User Guide](#), Using Subband-BTE Solver as an External Solver for Sentaurus Device, for details about using the Subband-BTE solver.

## Chapter 14: Quantization

### Density Gradient Model

## Density Gradient Model

For the density gradient model [3][4],  $\Lambda_n$  in [Equation 224](#) is given by a partial differential equation:

$$\Lambda_n = -\frac{\gamma \hbar^2}{12m_n} \nabla^2 \ln n + \frac{1}{2} (\nabla \ln n)^2 = -\frac{\gamma \hbar^2}{6m_n} \frac{\nabla^2 \sqrt{n}}{\sqrt{n}} \quad (234)$$

where  $\gamma = \gamma_0 \cdot \gamma_{\text{pmi}}$  is a fit factor. The  $\gamma_0$  is solution independent. It is dependent on the mole fraction and the distance to the nearest insulating surface (option `AutoOrientation`). The additional factor  $\gamma_{\text{pmi}}$  (solution dependent) can be defined by a PMI (see [Gamma Factor for Density Gradient Model on page 1457](#)).

Introducing the reciprocal thermal energy  $\beta = 1/kT_n$ , the mass-driving term  $\Phi_m = -kT_n \ln(N_C/N_{\text{ref}})$  (with an arbitrary normalization constant  $N_{\text{ref}}$ ) and the potential-like quantity  $\bar{\Phi} = E_C + \Phi_m + \Lambda_n$ , [Equation 234](#) can be rewritten and generalized as:

$$\begin{aligned} \Lambda_n = & -\frac{\hbar^2 \gamma}{12m_n} \{ \nabla \cdot \alpha (\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \\ & + \vartheta (\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \cdot \alpha (\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \} + \Lambda_{\text{PMI}} \end{aligned} \quad (235)$$

with the default parameters  $\xi = \eta = 1$ ,  $\vartheta = 1/2$ , and  $\alpha$  is a symmetric matrix that defaults to one. In insulators, Sentaurus Device does not compute the Fermi energy; therefore, in insulators,  $\xi = \eta = 0$ . By default, Sentaurus Device uses [Equation 235](#), which for Fermi statistics deviates from [Equation 234](#), even when  $\xi = \eta = 1$  and  $\vartheta = 1/2$ . The density-based expression [Equation 234](#) is available as an option (see [Using the Density Gradient Model on page 363](#)).

Optionally, you can choose an alternative formula where the mass term appears *between* rather than *in front of* the differential operators:

$$\begin{aligned} \Lambda_n = & -\{ \nabla \cdot \frac{\hbar^2 \gamma}{12m_n} \alpha (\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \\ & + \vartheta (\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \cdot \frac{\hbar^2 \gamma}{12m_n} \alpha (\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \} + \Lambda_{\text{PMI}} \end{aligned} \quad (236)$$

While [Equation 235](#) is the default for reasons of backward compatibility, [Equation 236](#) resembles more closely the Schrödinger equation and, therefore, is considered physically superior.

In [Equation 235](#),  $\Lambda_{\text{PMI}}$  is a locally dependent quantity computed from a user-specified PMI model (see [Apparent Band-Edge Shift on page 1308](#)), from the Schenk bandgap narrowing model (see [Schenk Bandgap Narrowing Model on page 310](#)), or from apparent band-edge shifts caused by multistate configurations (see [Apparent Band-Edge Shift on page 595](#)). By default,  $\Lambda_{\text{PMI}} = 0$ .

## Chapter 14: Quantization

### Density Gradient Model

At Ohmic contacts, interfaces to metals with Ohmic boundary conditions, resistive contacts, and current contacts, the boundary condition  $\Lambda_n = \Lambda_{\text{PMI}}$  is imposed. At Schottky contacts, gate contacts, interfaces to metals with Schottky boundary conditions, and external boundaries, homogeneous Neumann boundary conditions are used:

$$\hat{n} \cdot \alpha(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) = 0.$$

At internal interfaces between regions where the density gradient equation is solved,  $\bar{\Phi}$  must be continuous. In addition, for [Equation 235](#),  $\hat{n} \cdot \alpha(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi)/m_n$  must be continuous.

For [Equation 236](#),  $\hat{n} \cdot \gamma \alpha(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi)/m_n$  must be continuous.

At internal interfaces between a region where the equation is solved (indicated below by  $0^-$ ) and a non-metal region where it is not solved ( $0^+$ ), an inhomogeneous Neumann boundary condition obtained from the analytic solution of the 1D step problem is applied:

$$\hat{n} \cdot \nabla \Lambda_n^+ = \sqrt{\frac{24m_n(0^+)kT_n}{\hbar^2\gamma(0^+)\bar{\alpha}}}[E_C(0^+) - E_C(0^-) - \Lambda_n]f \frac{2\vartheta(0^+)}{kT_n}[\Lambda_n - E_C(0^+) + E_C(0^-)] \quad (237)$$

where  $f(x) = \sqrt{[(\exp x - 1)/x - 1]/x}$  and  $\bar{\alpha} = \det[\alpha(0^+)]^{1/3}$ .

In conjunction with [Equation 237](#), a homogeneous Neumann boundary condition can be imposed on interfaces of regions where quantum effects are insignificant, that is, interfaces away from the channel region in MOS devices. You should always use [Equation 237](#) as the boundary condition on interfaces in the channel region where quantum effects can significantly alter charge carrier concentration. The homogeneous Neumann boundary condition is:

$$\hat{n} \cdot \nabla \Lambda_n^+ = 0 \quad (238)$$

Optionally, Sentaurus Device offers a modified mobility to improve the modeling of tunneling through semiconductor barriers:

$$\mu = \frac{\mu_{\text{cl}} + r\mu_{\text{tunnel}}}{1 + r} \quad (239)$$

where  $\mu_{\text{cl}}$  is the usual (classical) mobility as described in [Chapter 15 on page 385](#),  $\mu_{\text{tunnel}}$  is a fit parameter, and  $r = \max(0, n/n_{\text{cl}} - 1)$ . Here,  $n_{\text{cl}}$  is the ‘classical’ density (see [Equation 420](#)). In this modification, for  $n > n_{\text{cl}}$ , the additional carriers (density  $n - n_{\text{cl}}$ ) are considered as tunneling carriers that are subject to a different mobility  $\mu_{\text{tunnel}}$  than the classical carriers (density  $n_{\text{cl}}$ ).

## Using the Density Gradient Model

The density gradient equation for electrons and holes is activated by the `eQuantumPotential` and `hQuantumPotential` switches in the `Physics` section. Use

## Chapter 14: Quantization

### Density Gradient Model

-eQuantumPotential and -hQuantumPotential to switch off the equations. These switches can also be used in regionwise or materialwise Physics sections. In metal regions, the equations are never solved. For a summary of available options, see [Table 299 on page 1676](#).

The option `Formula` selects either [Equation 235](#) (`Formula=0`, default) or [Equation 236](#) (`Formula=1`).

The `Ignore` switch to `eQuantumPotential` and `hQuantumPotential` instructs Sentaurus Device to compute the quantum potential, but not to use it. `Ignore` is intended for backward compatibility to earlier versions of Sentaurus Device.

The `Resolve` switch enables a numeric approach that handles discontinuous band structures, at moderately refined interfaces that are not heterointerfaces, more accurately than the default approach.

The switch `Density` to `eQuantumPotential` and `hQuantumPotential` activates the density-based formula [Equation 234](#) instead of the potential-based formula [Equation 235](#). If this option is used, the parameters  $\xi$  and  $\eta$  must both be 1.

The `LocalModel` option of `eQuantumPotential` and `hQuantumPotential` specifies the name of a PMI model (see [Chapter 39 on page 1207](#)) or the Schenk bandgap narrowing model (see [Schenk Bandgap Narrowing Model on page 310](#)) for  $\Lambda_{\text{PMI}}$  in [Equation 235](#). An additional apparent band-edge shift contribution can be added if the corresponding model of a multistate configuration is switched on (see [Apparent Band-Edge Shift on page 595](#)).

The `BoundaryCondition` option of the `eQuantumPotential` and `hQuantumPotential` specifications in metal interface-specific or electrode-specific Physics sections allows you to explicitly specify the boundary condition for the quantum potential, overriding the default boundary condition. Possible values are `Dirichlet` and `Neumann`, to enforce homogeneous Dirichlet and Neumann boundary conditions, respectively.

The homogeneous Neumann boundary condition ([Equation 238](#)) can be specified in the command file in the following ways:

- Doping density-based specification in the `Math` section:

- `RefNd_QuantumPotential=<float>`

All vertices on semiconductor-insulator interfaces that have a semiconductor donor doping density greater than `RefNd_QuantumPotential` are treated with the homogeneous Neumann boundary condition.

- `RefNa_QuantumPotential=<float>`

All vertices on semiconductor-insulator interfaces that have a semiconductor acceptor doping density greater than `RefNa_QuantumPotential` are treated with the homogeneous Neumann boundary condition.

## Chapter 14: Quantization

### Density Gradient Model

For example:

```
Math {  
    RefNd_QualumPotential=<float>  
    RefNa_QualumPotential=<float>  
}
```

- Region interface-based specification in the `Physics` section:

The `BoundaryCondition` attribute of `e|hQuantumPotential` specifications in `Physics` sections allows you to specify the boundary condition for the quantum potential on various interfaces, overriding the default nonhomogeneous boundary condition in [Equation 237](#). For example:

```
Physics(RegionInterface="Region_1/Region_2") {  
    eQuantumPotential(BoundaryCondition = HomogeneousNeumann)  
}
```

The region interface specification can be combined with a doping-based specification in the `Math` section.

- Homogeneous boundary condition in a box:

In this approach, you use a box to apply a nonhomogeneous boundary condition only on interfaces inside the box. The box information is the same as that of `eQPBox` and `hQPBox` in the command file as described in [Notes on the Use of the Density Gradient Model on page 367](#) with the additional `QboxForBC` keyword. For example:

```
Math {  
    eQPBox(QboxForBC {MinX=0.042 MaxX=0.045 MinY=0.008MaxY=0.015}  
           {MinX=0.022 MaxX=0.027 MinY=0.002MaxY=0.004})  
    hQPBox(QboxForBC {MinX=0.042 MaxX=0.045 MinY=0.008MaxY=0.015}  
           {MinX=0.022 MaxX=0.027 MinY=0.002MaxY=0.004})  
}
```

Unlike the evaluation box described in [Notes on the Use of the Density Gradient Model](#), which effectively suppresses the quantum potential outside the box, the presence of the `QboxForBC` keyword allows for the density-gradient solution in all materials specified in the command file with an interface-specific boundary condition, that is, interfaces with nonhomogeneous boundary conditions inside the box and homogeneous boundary conditions outside the box.

Apart from activating the equations in the `Physics` section, the equations for the quantum corrections must be solved by using `eQuantumPotential` or `hQuantumPotential`, or both in the `Solve` section. For example:

```
Physics { eQuantumPotential }  
  
Plot { eQuantumPotential }
```

## Chapter 14: Quantization

### Density Gradient Model

```
Solve {
    Coupled { Poisson eQuantumPotential }
    Quasistationary (
        DoZero InitialStep=0.01 MaxStep=0.1 MinStep=1e-5
        Goal { Name="gate" Voltage=2 }
    ) {
        Coupled { Poisson Electron eQuantumPotential }
    }
}
```

The quantum corrections can be plotted. To this end, use `eQuantumPotential`, or `hQuantumPotential`, or both in the `Plot` section.

To activate the mobility modification according to [Equation 239](#), specify the `Tunneling` switch to `Mobility` in the `Physics` section of the command file. Specify  $\mu_{\text{tunnel}}$  for electrons and holes by the `mutunnel` parameter pair in the `ConstantMobility` parameter set.

The parameters  $\gamma$ ,  $\vartheta$ ,  $\xi$ , and  $\eta$  are available in the `QuantumPotentialParameters` parameter set. The parameters can be specified regionwise and materialwise. They cannot be functions of the mole fraction in heterodevices. In insulators,  $\xi$  is always assumed as zero, regardless of user specifications.

The diagonal of  $\alpha$  in the crystal coordinate system (see [Crystal and Simulation Coordinate Systems on page 886](#)) is determined by the parameter pairs `alpha[1]`, `alpha[2]`, and `alpha[3]` (for the x-direction, y-direction, and z-direction, respectively) in the `QuantumPotentialParameters` parameter set. Unless  $\alpha$  is a multiple of the unit matrix, Sentaurus Device solves an anisotropic problem.

The following example decreases the quantization effect along the y-direction to half, for both electrons and holes:

```
QuantumPotentialParameters {
    alpha[1] = 1   1
    alpha[2] = 0.5 0.5
}
```

The anisotropic density-gradient equation supports the `AverageAniso` approximation (see [Chapter 28 on page 883](#)) and the tensor grid approximation (see [Tensor Grid Option on page 999](#)). For the latter, `TensorGridAniso` must be selected as the numeric method in the `Math` section, and either `eQuantumPotential` or `hQuantumPotential` must be specified as an option of `Aniso` in the `Physics` section. For example:

```
Physics { Aniso (eQuantumPotential) }
Math { TensorGridAniso(Aniso) }
```

## Chapter 14: Quantization

### Density Gradient Model

## Named Parameter Sets for the Density Gradient Model

The `QuantumPotentialParameters` parameter set can be named. For example, in the parameter file, to declare a parameter set with the name `myset`, you can write:

```
QuantumPotentialParameters "myset" { ... }
```

To use a named parameter set, specify its name with `ParameterSetName` as an option to either `eQuantumPotential` or `hQuantumPotential` in the command file. For example:

```
Physics { eQuantumPotential (ParameterSetName = "myset") }
```

By default, the unnamed parameter set is used.

## Auto-Orientation for the Density Gradient Model

The `eQuantumPotential` and `hQuantumPotential` models support the auto-orientation framework (see [Auto-Orientation Framework on page 86](#)) that switches between different named parameter sets based on the orientation of the nearest interface. This can be activated by specifying `AutoOrientation` as an argument to either `eQuantumPotential` or `hQuantumPotential` in the command file. For example:

```
Physics { eQuantumPotential (AutoOrientation) }
```

---

## Notes on the Use of the Density Gradient Model

Note that:

- Fitting parameters: The parameter  $\gamma$  has been calibrated only for silicon. The quantum correction affects the densities and field distribution in a device. Therefore, parameters for mobility and recombination models that have been calibrated to classical simulations (or simulations with the van Dort model) might require recalibration.
- Evaluation box: You can improve convergence by solving the density gradient equations only within regions that are adequately meshed for that task. For example, in FinFETs, it is understood that, for a fin situated on a large substrate, given the oppressive cost of meshing the surface of the substrate to density gradient standards, it is beneficial to solve the equations only in the fin.

This works well if the fin can be specified as a unique region that coincides with the domain over which the quantum potential is useful. However, in structures formed by Sentaurus Process, such precise registration of regions might not be possible.

An alternate approach is to reduce the quantum potential in parts of a region that are not of interest. In this approach, the parameter  $\gamma$  effectively attenuates the quantum potential outside the 'Box' where the solution of quantum potential might cause convergence issues.

## Chapter 14: Quantization

### Density Gradient Model

The corresponding command file can be written as:

```
Math {
    eQPBox({MinX=0.042 MaxX=0.045 MinY=0.008MaxY=0.015
            Attenuation_Length=1e-3}
           {MinX=0.022 MaxX=0.027 MinY=0.002MaxY=0.004
            Attenuation_Length=2e-3})
    )
    hQPBox({MinX=0.042 MaxX=0.045 MinY=0.008MaxY=0.015
            Attenuation_Length=2e-3}
           {MinX=0.022 MaxX=0.027 MinY=0.002MaxY=0.004
            Attenuation_Length=4e-3})
    )
}
```

The parameter  $\gamma$  is effectively attenuated using an error function for each of the six specified domain boundaries. `Attenuation_Length` determines the decay of the transition, where 0 is abrupt, and the default value is  $1e-3 \mu\text{m}$ .

- Tunneling: The density gradient model increases the current through the semiconducting potential barriers. However, this effect is not a trustworthy description of tunneling through the barrier. To model tunneling, use one of the dedicated models that Sentaurus Device provides (see [Chapter 24 on page 819](#)). To suppress unwanted tunneling or to fit tunneling currents despite these concerns, consider using the modified mobility model according to [Equation 239](#).
- Convergence: In general and particularly for the density gradient corrections, solving additional equations worsens convergence. Typically, it is advisable to solve the equations for the quantum potentials whenever the Poisson equation is solved (using a `Coupled` statement). Usually, the best strategy to obtain an initial solution at the beginning of the simulation is to do a coupled solve of the Poisson equation and the quantum potentials, without the current and temperature equations.

To obtain this initial solution, it is sometimes necessary that you set the `LineSearchDamping` optional parameter (see [Damped Newton Iterations on page 195](#)) to a value less than 1; a good value is 0.01.

Often, using initial bias conditions that induce a current flow work well for a classical simulation, but do not work when the density gradient model is active. In such cases, start from equilibrium bias conditions and put an additional voltage ramping at the beginning of the simulation.

If an initial solution is still not possible, consider using Fermi statistics (see [Fermi Statistics on page 233](#)). Check grid refinement and pay special attention to interfaces between insulators and highly doped semiconductor regions. For classical simulations, the refinement perpendicular to such interfaces is often not critical, whereas for quantum mechanical simulations, quantization introduces variations at small length scales, which must be resolved.

## Chapter 14: Quantization

### Density Gradient Model

- Avalanche: The combination of using the density gradient model with an avalanche model can sometimes degrade convergence. An option is available in the `Math` section that can often result in better convergence in this situation:

```
Math {RefDens_QuantumPotential=<n0>}
```

where  $n_0$  is the carrier concentration below which the effects of quantum corrections are reduced.

- Speed: Activate the equations only for regions where the quantum corrections are physically needed. For example, usually, the density gradient equations do not need to be computed in insulators. Using the model selectively, typically, also benefits convergence.
- By default, the DOS mass used in the expressions of the density gradient method ignores the effects of strain (see [Strained Effective Masses and Density-of-States on page 945](#)). This is accomplished by using the expression  $m_n = m_{n0} + v \cdot \Delta m_n$  with a default value of  $v = 0$ , where  $m_{n0}$  is the unstrained mass and  $\Delta m_n$  is the change in mass due to strain. The parameter  $v$  can be specified in the `QuantumPotentialParameters` parameter set.
- Homogeneous Neumann boundary condition: Applying this boundary condition ([Equation 238](#)) to interfaces in highly doped regions and to semiconductor–shallow trench isolation interfaces often results in better convergence behavior without affecting device characteristics. It is particularly helpful when impact ionization and nonlocal band-to-band tunneling models are used for device simulations (see [Chapter 16 on page 473](#)).
- By default, the density gradient model uses an exponential approximation in the continuity equation, which gives its solution as the carrier concentration multiplied by a quantum-mechanical factor. To activate another approximation where the quantum potential is applied directly to the electrostatic potential as a quantum-mechanical band edge, use keyword `DirectQuantumCorrection` in the global `Math` section.
- The default usage of the density gradient model (without `DirectQuantumCorrection` in the `Math` section) results in two solutions for the carrier density: classical and quantum mechanical. Usually, the `eDensity` and `hDensity` used directly in the `Plot` and `CurrentPlot` sections output the quantum-mechanical density. However, when implementing a `CurrentPlot` Tcl formula (see [Tcl Formulas on page 170](#)) or using a PMI model implementation (see [Introduction to the Physical Model Interface on page 1207](#)), you should explicitly distinguish between classical and quantum-mechanical datasets, where the application of `eQMDensity` and `hQMDensity` will use the quantum-mechanical density, and the application of `eDensity` and `hDensity` will use the classical density in the corresponding computations. For other quantum-correction options, both `eDensity` and `eQMDensity` (and similarly `hDensity` and `hQMDensity`) give equivalent carrier density profiles.

## Modified Local-Density Approximation Model

The modified local-density approximation (MLDA) model is a quantum-mechanical model that calculates the confined carrier distributions that occur near semiconductor–insulator interfaces [5]. It can be applied to both inversion and accumulation, and simultaneously to electrons and holes. It is based on a rigorous extension of the local-density approximation and provides a good compromise between accuracy and runtime.

Following Paasch and Übensee [5], the confined electron density at a distance  $z$  from a Si– $\text{SiO}_2$  interface is given, under Fermi statistics, by:

$$n_{\text{MLDA}}(\eta_n) = N_C \frac{2}{\sqrt{\pi}} \sum_0^{\infty} \int_{\epsilon}^{\infty} \frac{\sqrt{\epsilon}}{1 + \exp[(\epsilon - \eta_n)]} [1 - j_0(2z\sqrt{\epsilon}/\lambda_n)] d\epsilon \quad (240)$$

where:

- $\eta_n$  is given by [Equation 49 on page 233](#).
- $j_0$  is the 0th-order spherical Bessel function.
- $\lambda_n = \sqrt{\hbar^2/2m_{qn}kT_n}$  is the electron thermal wavelength which depends on the quantization mass  $m_{qn}$ .

The integrand of [Equation 240](#) is very similar to the classical Fermi integrand, with an additional factor describing confinement for small  $z$ . A similar equation is applied to holes.

Sentaurus Device provides the following MLDA model options:

- One option is the simple original model that uses only one user-defined parameter  $\lambda$  per carrier type.
- The second option (considered in [Interface Orientation and Stress Dependencies](#)) accounts for the multivalley/band property of the electron and hole band structures.

For the simple model, in Boltzmann statistics, [Equation 240](#) simplifies to the following expression for the electron density (the hole density is similar):

$$n_{\text{MLDA}}(\eta_n) = N_C \exp(\eta_n) [1 - \exp(-(z/\lambda_n)^2)] \quad (241)$$


---

## Interface Orientation and Stress Dependencies

The MLDA model allows you to consider a dependency of the quantization effect on interface orientation and stress. Mainly, such dependencies come from mass anisotropy and stress related valley/band energy change.

Mass anisotropy has been considered in the literature [6] where ellipsoidal type of bands, which correspond to three electron  $\Delta_2$ -valleys in silicon, were used. The main result of the

## Chapter 14: Quantization

### Modified Local-Density Approximation Model

article is that [Equation 240](#) is still valid for each of three electron  $\Delta_2$ -valleys with the effective DOS equal to  $N_C/3$ .

Another result is that the quantization mass of each valley  $m_{qn}^i$  should be computed as a rotation of the inverse mass tensor from the crystal system (where the tensor is diagonal) to another one where one axis is perpendicular to the interface and the mass component, which corresponds to the axis, should be taken as  $m_{qn}^i$ .

Considering three electron  $\Delta_2$ -valleys, [Equation 240](#) could be rewritten as follows:

$$n_{\text{MLDA}}(\eta_n, z) = \sum_{i=1}^{3 \infty} \frac{D_n^i(\epsilon, z)}{1 + \exp(\epsilon - \eta_n - \Delta\eta^i)} d\epsilon \quad (242)$$

$$D_n^i(\epsilon, z) = N_C g_n^i \frac{2}{\sqrt{\pi}} \sqrt{\epsilon} \left[ 1 - j_0 \frac{2z}{\sqrt{\frac{2m_{qn}^i k T_n \epsilon}{\hbar^2}}} \right]$$

where:

- $D_n^i(\epsilon, z)$  is the electron DOS of valley  $i$  with a dependency on the normalized energy  $\epsilon$  and on the distance to the interface  $z$ .
- $g_n^i$  is defined by [Equation 208](#) or [Equation 209](#); however, in the case of stress,  $N_C g_n^i$  is defined by [Equation 1006](#).
- $\Delta\eta^i$  represents the stress-induced valley energy change (see [Deformation of Band Structure on page 940](#)).

Similar to [Equation 242](#), the same multivalley MLDA quantization model could be applied to holes with hole bands defined in the `MultiValley` section of the parameter file described in [Multivalley Band Structure on page 327](#), but such spherical or ellipsoidal hole bands do not produce a correct dependency of the hole quantization on the interface orientation. Therefore, for holes, the six-band  $k \cdot p$  band structure is used. The bulk six-band  $k \cdot p$  model is described in [14][15] of [Chapter 31 on page 935](#). An extension of the MLDA model is formulated for arbitrary bands and applied to the six-band  $k \cdot p$  band structure [7].

The DOS of one hole band is written generally as follows:

$$D^i(\epsilon, z) = \frac{2}{(2\pi)^3} \circ \frac{dS}{|\nabla_k \epsilon^i(\vec{k})|} \left[ 1 - \exp \left( i2z\gamma_{kp} \int_j \frac{w_{j3}(\vec{k})}{w_{33}(\vec{k})} \right) \right] \quad (243)$$

where:

- $\epsilon^i(\vec{k})$  is a hole band dispersion of the bulk six-band  $k \cdot p$  model (the model considers three bands: heavy holes, light holes, and the spin-orbit split-off band holes).
- The surface integral is in  $k$ -space over isoenergy surface  $S$  defined by  $\epsilon^i(\vec{k}) = \epsilon$ .

## Chapter 14: Quantization

Modified Local-Density Approximation Model

- $w_{ij}$  are components of the reciprocal mass tensor computed locally on the isoenergy surface.
- $\gamma_{kp}$  is a fitting parameter that accounts for the nonparabolicity of hole bands in the MLDA model.

The hole density near the interface is computed similarly to the electron one in [Equation 242](#), but with the DOS from [Equation 243](#). The interface orientation and stress dependencies of the hole quantization are defined naturally by the physical property of the six-band k·p model.

## Heterojunctions

Originally, the MLDA quantization model was developed to describe the carrier density at semiconductor–insulator interfaces where a zero density (wavefunction) assumption on the interface is a reasonable approximation due to a typically high-band offset between the semiconductor and insulator. Such an assumption is mostly incorrect if the model is applied to heterojunctions where the band offset between two semiconductors is small.

To apply the MLDA model to heterojunctions [\[8\]](#), you can use a wave penetration length in the model:

$$z_0 = \frac{\hbar}{\sqrt{2m\Delta\varepsilon}} \quad (244)$$

where  $m$  is the carrier (electron or hole) effective mass of the low-bandgap semiconductor, and  $\Delta\varepsilon$  is the band offset between two semiconductors (in the conduction or valence band). According to [\[8\]](#), the penetration length  $z_0$  is simply added to the interface distance  $z$  in [Equation 240](#)–[Equation 243](#).

## Nonparabolic Bands and Geometric Quantization

Other effects that can play an important role, for example, in III–V semiconductors are the band nonparabolicity and geometric quantization in thin-layer structures. For the bulk case, the carrier density computation with the nonparabolic bands is described in [Nonparabolic Band Structure on page 328](#). To account for this effect and 1D geometric quantization (between two parallel interfaces that confine the carriers) in the MLDA model, the DOS in [Equation 242](#) must be replaced with the following [\[9\]](#):

$$D_n^i(\varepsilon, z) = N_C g_n^i \frac{2}{\sqrt{\pi}} (\frac{1}{2} + \frac{2kT_n\alpha^i \varepsilon'}{\hbar^2}) \sqrt{\varepsilon'(1 + kT_n\alpha^i \varepsilon')} [1 - j_0(zK) - j_0((L_z - z)K) + j_0(L_z K)]$$

$$K = 2 \sqrt{\frac{2m_{qn}^i k T_n \varepsilon' (1 + k T_n \alpha^i \varepsilon')}{\hbar^2}} \quad \varepsilon' = \varepsilon + \frac{\varepsilon_1}{k T_n} \quad \varepsilon_1 = \frac{-1 + \sqrt{1 + \frac{2\alpha^i \hbar \pi}{m_{qn}^i L_z}^2}}{2\alpha^i} \quad (245)$$

## Chapter 14: Quantization

### Modified Local-Density Approximation Model

where:

- $\alpha^i$  is the band nonparabolicity.
- $L_z$  is a distance between the parallel interfaces.
- $\varepsilon_1$  is the first subband energy of the infinite-barrier quantum well with size  $L_z$ .

[Equation 245](#) assumes that the DOS is equal to zero up to the first subband energy  $\varepsilon_1$  and this gives an effect of bandgap widening in thin-layer structures.

For the hole six-band  $k \cdot p$  bands in [Equation 243](#), geometric quantization is accounted for similarly by using the first subband energy  $\varepsilon_1$ . In this case, the quantization mass in  $\varepsilon_1$  is taken to be energy dependent along the quantization direction, which comes from the reciprocal mass tensor  $w_{ij}$  used in [Equation 243](#).

This MLDA quantization option is implemented using the multivalley carrier density model described in [Multivalley Band Structure on page 327](#). It uses all of the valley and mass parameter definitions of the multivalley model in the `MultiValley` section of the parameter file. In addition, it applies a numeric integration of [Equation 240](#), [Equation 242](#) based on Gauss–Laguerre quadratures as described in [Nonparabolic Band Structure on page 328](#). The numeric integration is applied to both Fermi–Dirac statistics and Boltzmann statistics. By default, the model automatically finds the closest interface and accounts for its orientation by recomputing the quantization mass for each valley and channel mesh point. The interface orientation is found using a vector normal to the interface and an orientation of the simulation coordinate system in reference to the crystal system defined in the `LatticeParameters` section of the parameter file (see [Crystal and Simulation Coordinate Systems on page 886](#)). For fitting purposes, you have an option to define a specific quantization mass for each valley using the parameter file (see [Using the MLDA Model](#)).

---

## Using the MLDA Model

To activate the multivalley orientation-dependent model, the keyword `MLDA` must be used in the `MultiValley` statement of the `Physics` section:

```
Physics { MultiValley(MLDA) }
```

To activate the model only for electrons or holes, use `eMultiValley(MLDA)` or `hMultiValley(MLDA)`, respectively.

You can modify parameters of the model in the `MultiValley` section of the parameter file (see [Using Multivalley Band Structure on page 331](#)). By default, longitudinal and transverse effective masses are specified in the `(e|h)Valley` sections. These masses form the inverse mass tensor in the valley ellipsoidal coordinate system, which is rotated to the interface system, and is used to compute the quantization mass  $m_q$ .

## Chapter 14: Quantization

### Modified Local-Density Approximation Model

If you want to define a constant quantization mass per valley, ignoring automatic interface orientation dependency, there is an option to define it in the `Valley` statement:

```
MultiValley { (e|h)Valley(mq = mq)}
```

To activate the multivalley MLDA model based on the six-band  $k \cdot p$  band structure for holes (see [Equation 243](#)) or the two-band  $k \cdot p$  model for  $\Delta_2$  electron valleys, an additional keyword must be used:

```
Physics { MultiValley(MLDA kpDOS) }
```

In this case, all  $k \cdot p$  model parameters are defined in the `LatticeParameters` section of the parameter file (see [Using Strained Effective Masses and DOS on page 949](#)). The fitting parameter  $\gamma_{kp}$  in [Equation 243](#) should be specified as a value of the parameter `hkpdosfactor` in the `MLDAQMModel` section of the parameter file as follows:

```
MLDAQMModel{  
    hkpdosfactor = 0.4  
    ekpdosfactor = 1.0  
}
```

The parameter `ekpdosfactor` can be used as a fitting parameter for the electron multivalley MLDA model (as a factor to  $z$  in [Equation 242](#)) if `eMultiValley(MLDA kpDOS)` is specified in the `Physics` section of the command file. To use the multivalley MLDA model for both  $k \cdot p$  bands and valleys defined in the parameter file, `MultiValley(MLDA kpDOS parfile)` must be used.

To apply the MLDA model to heterojunctions, the penetration length  $z_0$  (see [Equation 244](#)) must be specified (default is zero) in the parameter file for electrons and holes separately. This length is considered to be a fitting parameter specific for heterojunctions, and [Equation 244](#) should be used only as a reference.

For example, with a 5 Å penetration, the penetration length can be specified as:

```
MLDAQMModel{  
    ePenetration = 5e-8    # [cm]  
    hPenetration = 5e-8    # [cm]  
}
```

To activate the multivalley MLDA model with nonparabolic bands ([Equation 245](#)), the keywords `Multivalley(MLDA Nonparabolicity)` must be specified. If the nonparabolicity should be ignored only in the MLDA part of the model (in the Bessel functions  $j_0$  of [Equation 245](#)), you must use `MLDA(-Nonparabolicity)`.

To have the geometric quantization accounted for as in [Equation 245](#) (the part dependent on the quantization length  $L_z$ ), the `ThinLayer` keyword must be in the `MultiValley` statement and the `LayerThickness` statement must be set. All `LayerThickness` options are described in [Extracting Layer Thickness on page 379](#) where generally the layer thickness (the quantization length  $L_z$ ) can be extracted automatically for defined regions.

## Chapter 14: Quantization

### Modified Local-Density Approximation Model

Practically, if the geometric quantization is mostly 1D, such a setting can be as simple as:

```
LayerThickness( Thickness = LZ)
Multivalley(MLDA ThinLayer)
```

For more complicated cases, such as for rectangular nanowires, the geometric quantization at corners is important and the following statements with automatic extraction of the quantization length  $L_z$  can be used:

```
LayerThickness( MaxFitWeight = 0.35 DimensionWeight = 1.0 )
Multivalley(MLDA ThinLayer)
```

The parameter `DimensionWeight` is specific to the model where the first subband energy  $\epsilon_1$  is used, as in [Equation 245](#). This parameter works as a factor to  $\epsilon_1$  and is designed as a fitting factor to effectively account for multidimensional quantization. Its default value is 1, but it should be increased with a reduction of the nanowire size.

To plot the electron and hole effective quantum corrections of the quantization models activated in the `Multivalley` statement (`MLDA` and `ThinLayer`), the `eMVQMBandgapShift` and `hMVQMBandgapShift` keywords must be present in the `Plot` section.

By default, the `MLDA` model looks for all semiconductor–insulator interfaces and calculates the normal distance  $z$  in [Equation 240](#)–[Equation 243](#) to the nearest interface. The `MLDA` model also uses the `EnormalInterface` option and the `GeometricDistances` option (see [Normal to the Interface on page 426](#)). To improve numerics if there is a coarse mesh in highly doped regions, there is an option to compute an averaged distance from a vertex to the interface based on element volumes and interface distances of elements, which contain the vertex (surrounding elements).

The option is implemented only for the multivalley `MLDA` and has an adjusting parameter that can be specified as follows:

```
Math { MVMLDAcontrols(AveDistanceFactor = 0.05) }
```

The default value of the parameter is 0.05 but, if it is equal to zero, the normal distance  $z$  is not modified by such averaging. If the parameter is unit, the distance  $z$  will be computed as an averaged distance using the normal distances of all vertices of the surrounded semiconductor elements with element–vertex volume weights.

To control a distance from the interface where the multivalley `MLDA` models will be applied, use the following statement:

```
Math { MVMLDAcontrols(MaxIntDistance = 1e-6) }
```

The default value of `MaxIntDistance` is  $10^{-6}$  cm.

To control the multivalley `MLDA` models by the doping concentration (for example, to exclude partially source/drain regions), the following command can be used:

```
Math { MVMLDAcontrols(MaxDoping4Majority = 1e19) }
```

## Chapter 14: Quantization

### Modified Local-Density Approximation Model

where `MaxDoping4Majority` defines the maximum doping concentration where the multivalley MLDA models are applied for majority carriers. For example, if `MaxDoping4Majority` is equal to zero, it excludes source/drain regions completely and the models are applied only for minority carriers. The default value of `MaxDoping4Majority` is  $10^{22} \text{ cm}^{-3}$ , that is, there are no limitations related to the doping concentration.

If the statement `hMultivalley(MLDA kpDOS)` is activated, then the model performs an initial computation of energy-dependent data based on the bulk six-band  $k \cdot p$  dispersion model. This computation might be time consuming, especially for 3D simulations and with the `hSubband` model active (see [Mobility Modeling on page 952](#)).

To control the accuracy of surface integration in [Equation 243](#), you can use the following grid setting in  $k$ -space, `Math{ kGridSizeFor6kpMLDA(Nphi,Ntheta) }`, where `Nphi` and `Ntheta` define the number of points for an angular spherical coordinate system. Due to symmetry and possibly the stress effect in DOS, half of the  $k$ -space is used, where both angles are changed from  $0^\circ$  to  $180^\circ$ . The default setting is `kGridSizeFor6kpMLDA(25, 25)`.

To separate this initial computation from bias ramping, there is an option to save and load the energy-dependent data file. For example:

```
Math { MVMLDAcontrols(Save = "file_name") }
Math { MVMLDAcontrols(Load = "file_name") }
```

If `LoadWithInterpolation= "file_name"` is used, then the saved data file could have a mesh different to the simulation one, and this option will interpolate the data to the simulation mesh.

In addition, to exclude source/drain or other regions, the keyword `MLDABox` can be used to define the range of the interface from which the MLDA distance function is calculated:

```
Math{
    EnormalInterface(
        regionInterface=[ "SemiRegion/InsulRegion" ]
        materialInterface=[ "OxideAsSemiconductor/Silicon" ]
    )
    * single MLDABox
    MLDABox( MinX=-0.1, MaxX=0.1, MinY=-0.01, MaxY=0.01 )
    * multiple MLDABox
    MLDABox( { MinX=-0.1, MaxX=0.1, MinY=-0.01, MaxY=0.01 }
              { MinX=-0.1, MaxX=0.1, MinY= 0.05, MaxY=0.06 } )
}
```

The unit of the parameters `MinX`, `MaxX`, `MinY`, and `MaxY` in the `MLDABox` is micrometer.

To activate the simple MLDA model (with one parameter  $\lambda$  per carrier type), use the keyword `MLDA` in the `Physics` section. To activate the model for electrons or holes only, use `eMLDA` or `hMLDA`. The model also can be specified in the regionwise or materialwise `Physics` section, and can be switched off by using `-MLDA`, `-eMLDA`, or `-hMLDA`.

## Chapter 14: Quantization

### Modified Local-Density Approximation Model

By default, the thermal wavelength is computed from the thermal wavelengths at 300 K as  $\lambda = \lambda_{300} \sqrt{300 \text{ K} / T_n}$ ; alternatively,  $\lambda = \lambda_{300}$  can be used.

To activate or deactivate the temperature dependence of the thermal wavelength, the LambdaTemp switch can be specified as an option to MLDA:

```
Physics {
    MLDA (
        -LambdaTemp      * eMLDA | hMLDA for one carrier
                           * switch off temperature dependency
    )
}
```

In Table 46, the defaults of  $\lambda_{300}$  were determined by comparison with the Schrödinger equation solver at 23.5 Å and 25.0 Å for electrons and holes, respectively. These parameters are accessible in the MLDA parameter set.

Table 46 Default coefficients for MLDA model

Symbol	Electrons	Holes	Unit
$\lambda_{300}$	eLambda	$23.5 \times 10^{-8}$	cm

To plot the electron and hole effective quantum corrections of the simple MLDA model (see Equation 240), the eMLDAQuantumPotential and hMLDAQuantumPotential keywords must be present in the Plot section.

---

## Notes on the Use of the MLDA Model

Note that:

- For each carrier type, only one of the two MLDA options (either multivalley ( $e|h$ )Multivalley(MLDA) or only ( $e|h$ )MLDA) can be used in the Physics section.
- The multivalley MLDA model can be used to compute stress-related mobility change (see Mobility Modeling on page 952) and, therefore, it is required to be switched on for these mobility models. However, for some cases, other quantum models might be needed in the carrier density computation. For such cases, the multivalley MLDA model can be activated for all models except the density as follows:

`(e|h)Multivalley(MLDA -Density)`

Such activation (exclusion of the multivalley option from the density computation) is global for the whole device even if it appears in one region of the device only.

- Theoretically, the MLDA model should give zero carrier density at the semiconductor-insulator interface. However, since the zero carrier density results in various numeric problems, the actual values of carrier density at the interface calculated by the program

are very small but greater than zero. This is achieved by assuming a very thin transition layer (one-hundredth of an ångström) between the insulator and the semiconductor.

- MLDA quantization is switched off at contacts and at a small distance to the contacts (to avoid numeric and physical inconsistencies). By default, the distance to the contacts (where the model is switched off) is 75 Å.

To change the distance, use `MLDAMinDistanceToContact=<number>` (in units of µm) in the `Math` section.

## Quantum-Well Quantization Model

The quantum-well quantization model is activated in the `Physics` section for semiconductor regions using the `eDensityCorrection` and `hDensityCorrection` options of `QWLocal`. For example:

```
Physics(Region = "well") {
    QWLocal(
        eDensityCorrection
        hDensityCorrection
    )
}
```

For additional options of `QWLocal`, see [Localized Quantum-Well Model on page 1092](#) and [Table 256 on page 1647](#).

The *quantum well* is composed of all regions in which this model is active. Sentaurus Device computes the local thickness  $t$  of the quantum well and, together with the local electric field  $F$ , computes the electron density as:

$$n = \frac{kT}{t\hbar^2\pi} \sum_v d_v m_{xy,v} F_0 \frac{\tilde{E}_{F,n} - \tilde{E}_C - E_{j,v}(F, t)}{kT} \quad (246)$$

A similar expression is used for holes. In [Equation 246](#),  $j$  runs over all subbands of band  $v$ . The eigenenergy  $E_{j,v}(F, t)$  is measured relative to the band edge  $E_C$  at the center of the well.  $E_{j,v}(F, t)$  and  $E_C$  are computed from the local field  $F$  and the local band edge  $E_C$ , assuming that the field in the well is constant and equal to  $F$ , and that the barriers limiting the well are symmetric and are given by the average barrier height of the entire quantum well.

During Newton iterations, large fields can occur, which can cause the `QWLocal` implementation to take a lot of time and memory. With the `MaxElectricField` parameter of `QWLocal`, you can set a cutoff value for the field. The default is  $10^6$  V/cm. To ensure that the cutoff does not affect the result, you can check that, for the converged solution, the electric field in the well does not exceed the cutoff.

## Chapter 14: Quantization

### Extracting Layer Thickness

Contrary to the 1D Schrödinger equation (see [Parameters of the 1D Schrödinger Equation on page 352](#)), which can extract the number of ladders and its parameters automatically, an explicit ladder specification (see [Explicitly Specifying Ladders on page 353](#)) is required in the parameter file. The number of bands in the ladders must be the same over the entire quantum well and in the regions adjacent to it. To specify the number of subbands to be computed for electrons, heavy holes, and light holes, use the keywords `NumberOfElectronSubbands`, `NumberOfHeavyHoleSubbands`, and `NumberOfLightHoleSubbands`, respectively.

#### Note:

To use the quantum-well quantization model, you must use the `HeteroInterface` or the `Thermionic` keyword. Furthermore, each quantum-well region must be adjacent to at least one semiconductor barrier region.

To use the quantum-well quantization model in the context of the simulation of light-emitting diodes, see [Localized Quantum-Well Model on page 1092](#) and [Table 256 on page 1647](#).

---

## Extracting Layer Thickness

Sentaurus Device provides models (for example, MLDA and mobility) that account for quantum-mechanical effects that occur in thin semiconducting films. These models use a parameter `LayerThickness`, which is intrinsically one-dimensional; it is well defined only in the case of an infinite sheet of fixed thickness. Sentaurus Device generalizes this parameter to arbitrary structures in two and three dimensions using the concept of the *radius of the largest sphere that fits in a material and touches the surface point nearest a given point*. Therefore, a `LayerThickness` is defined as a scalar field throughout a structure.

Sentaurus Device can compute `LayerThickness` in one of the following ways:

- Using the `LayerThickness` command:

```
LayerThickness(<geo_parameters>)
```

- Using the `ThinLayer` subcommand (see [Thin-Layer Mobility Model on page 427](#)):

```
Mobility(ThinLayer(<geo/physical_parameters>))
```

As a result, there are two scalar arrays: `LayerThickness` and `LayerThicknessField`.

By default, both the `LayerThickness` and `LayerThicknessField` arrays are valid only in semiconductors. For `LayerThicknessField` only, the material group where the array is valid can be selected by the `AllowLayerThickness` keyword in the global `Math` section. Possible values are `Semiconductor` (default), `Insulator`, `Metal`, and `Everywhere`.

## Chapter 14: Quantization

### Extracting Layer Thickness

#### Note:

Use the `LayerThickness` command only in region- or material-specific `Physics` sections. In particular, if you use the command in the global `Physics` section, the layer will be composed of all regions of the allowed material types. Using the `LayerThickness` command with `AllowLayerThickness=Everywhere` always results in an infinite layer.

---

## Combining LayerThickness Command and ThinLayer Subcommand

To activate the `LayerThickness` command, you must specify the `LayerThickness` keyword with geometric parameters in the region, or material, or global `Physics` section. If the mobility model contains the `ThinLayer` subcommand with geometric parameters, this model uses the `LayerThickness` array; otherwise, the `LayerThicknessField` array will be used. For all other `ThinLayer`-dependent models, an external `LayerThickness` command must be set.

### Example 1

```
Physics (Material= Silicon) {
    # LayerThickness command: computation LayerThicknessField array
    LayerThickness(<geo_params>

    # ThinLayer without <geo_params>: mobility uses LayerThicknessField
    # array
    Mobility(ThinLayer(<physical_params>))
    MultiValley( ThinLayer ) # MultiValley uses LayerThicknessField array
}
```

### Example 2

```
Physics (Material= Silicon) {
    # LayerThickness command: computation LayerThicknessField array
    LayerThickness(<geo_params>

    # ThinLayer with <geo-params>: mobility uses LayerThickness array
    Mobility( ThinLayer(<geo_params> <physical_params>))

    MultiValley( ThinLayer ) # MultiValley uses LayerThicknessField array
}
```

### Example 3

```
Physics (Material= Silicon) { # without external LayerThickness command
    # mobility uses LayerThickness array
    Mobility( ThinLayer(<geo_params> <physical_params>))
    MultiValley( ThinLayer ) # error: LayerThickness command must be set
}
```

---

## Geometric Parameters of LayerThickness Command

The `LayerThickness` command has the following optional parameters: `Thickness`, `ChordWeight`, `MinAngle`, and `MaxFitWeight`.

**Note:**

The `ThinLayer` subcommand has the same `<geo_params>` as well as the optional `<physical_params>` (see [Using the Thin-Layer Mobility Model on page 429](#)).

The parameter `Thickness` explicitly specifies the thickness of a layer (in micrometers). If it is not present, Sentaurus Device extracts the thickness automatically. For the extraction, it assumes that all regions where `ThinLayer` is specified (regardless of carrier type) belong to the thin layer.

**Note:**

The external boundary is not considered to be a boundary of the thin layer. However, each interface between a region that specifies `ThinLayer` and a region that does not is considered to be a boundary of the thin layer, even if these two regions are both semiconductor regions.

The following example activates the computation of the `LayerThicknessField` array in regions `A` and `B`:

```
Physics(Region="A") { LayerThickness() }
Physics(Region="B") { LayerThickness(Thickness=0.005) }
```

Its thickness is extracted in region `A` and is assumed to be 5 nm in region `B`.

If the structure is a part of a larger symmetric structure and the full structure can be obtained by mirroring the simulated structure at a symmetry plane, the option `Mirror` must be activated in the global `Math` section.

`Mirror` is a vector that has the dimension of the device. Each component of the vector denotes the mirroring property for the corresponding axis and can have one of the values:

- `None` (no symmetry plane perpendicular to the axis)
- `Max` (symmetry plane at the largest coordinate of the axis)
- `Min` (symmetry plane at smallest coordinate)
- `Both` (symmetry planes at largest and smallest coordinates)

## Chapter 14: Quantization

### Extracting Layer Thickness

The following example means that there is a symmetry plane at the largest y-coordinate, as well as the smallest and largest z-coordinates:

```
Math {
    ThinLayer ( Mirror = (None Max Both) )
}
```

The full structure is six times bigger than the simulated one.

You can verify the proper thickness extraction using the plot variables `LayerThickness` and `LayerThicknessField`. Note that the layer thickness is defined in such a way that it becomes zero in concave corners of the layer. In addition, due to interpolation, `LayerThickness` can deviate by approximately half of the local mesh spacing from the thickness actually used for computation. For more information on thickness extraction and the options to control it, see [Thickness Extraction](#).

## Thickness Extraction

The definition of the *thickness* of a layer in a general geometry is not self-evident. This section describes how Sentaurus Device defines thickness.

To determine the thickness of a layer at a given point  $P$ , Sentaurus Device first finds the closest point  $C_1$  on the surface of the layer. Then, a sphere is constructed that touches the surface in  $C_1$  and has its midpoint within the layer, on the line passing through  $C_1$  and  $P$ .

Then, the diameter of this sphere is increased, until it touches the surface at a second point,  $C_2$ , and the surface normal at this point encloses an angle with the vector pointing from  $C_1$  to  $P$  of at least  $\beta$ . By default,  $\beta = 0$ , so any second touch point  $C_2$  is accepted, and the construction results in a sphere that is completely within the layer.

The thickness is a linear combination of the diameter of the final sphere,  $d$ , and the distance  $c_{12}$  between  $C_1$  and  $C_2$  (the chord length):

$$t = \alpha c_{12} + (1 - \alpha)d \quad (247)$$

The parameter  $\alpha$  can be set with the keyword `ChordWeight`, which is an option of `ThinLayer`. The default value is zero.

The value of angle  $\beta$  can be changed with the keyword `MinAngle`, an option to `ThinLayer`. `MinAngle` expects two values: The first value sets  $\beta$ , and the second value sets an angle  $\gamma$  that must be larger than  $\beta$ .

If the angle of the surface normal at  $C_2$  and the vector pointing from  $C_1$  to  $P$  is between  $\beta$  and  $\gamma$ , the value of the diameter  $d$  is multiplied by a factor that is very large if the angle is close to  $\beta$ , and by a factor that approaches one when the angle approaches  $\gamma$ . The purpose of the transition region between  $\beta$  and  $\gamma$  is to smooth the transition between touch points that fulfill the angle criterion and points that do not.

## Chapter 14: Quantization

### Extracting Layer Thickness

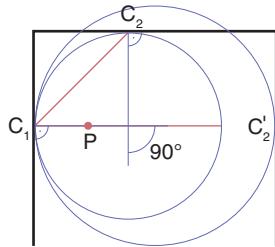
The following example sets  $\alpha$  to 0.5,  $\beta$  to  $89^\circ$ , and  $\gamma$  to  $90^\circ$ :

```
LayerThickness(  
    MinAngle=(89, 90)  
    ChordWeight=0.5  
)
```

The default value for  $\beta$  is zero, which causes the extracted layer thickness to go to zero in concave corners of the layer, even when the corners have a wide opening angle. This can be unwanted, and choosing a larger value for  $\beta$  can resolve this issue. However, there can be situations where a larger  $\beta$  can make the thickness extraction ambiguous, for example, when the choice of  $C_1$  is not unique.

To illustrate the thickness extraction, consider the example in [Figure 19](#).

*Figure 19* Thickness extraction



The thick black line is the boundary of the thin layer, and the red dot  $P$  indicates the position at which the thickness is extracted. First, the closest boundary point  $C_1$  is found, which is to the left of  $P$ . Then, with the default  $\beta = 0$ , the largest circle touching at  $C_1$  is inscribed into the layer. This is the smaller circle, with a second touch point  $C_2$ . From this circle, the layer thickness is obtained by a linear combination of the chord length (diagonal red line) and the diameter (horizontal red line).

The angle between the normal vectors  $C_1$  and  $C_2$  is  $90^\circ$ . Therefore, if the value of  $\beta$  is greater than  $90^\circ$  but smaller than  $180^\circ$ , the smaller circle is not considered. Instead, the larger circle (with a second touch point  $C'_2$ ) determines the layer thickness; for this circle, the angle between the normal vectors is  $180^\circ$ .

There is another option to compute the layer thickness, which also is based on considered spheres (or circles in [Figure 19](#) for the 2D case), but with some of the following modifications. The above algorithm for each point  $P$  (that is, for each mesh element center) constructs a sphere that satisfies the described conditions. As a result, you have a set of spheres that cover the region where the thickness should be computed. After that, for each mesh element (the point  $P$ ), you check whether this point is inside some subset of spheres, and you take the maximum sphere diameter  $d_m$  from this subset as the thickness for the point  $P$ . This way might be useful for spherical, circular, or nonplanar boundaries.

## Chapter 14: Quantization

### References

The user-defined keyword `MaxFitWeight` controls this option as follows:

$$t_m = \alpha_m d_m + (1 - \alpha_m)t \quad (248)$$

where  $t$  is the thickness from [Equation 247](#), and  $\alpha_m$  is defined by the keyword `MaxFitWeight` and is equal to zero by default.

---

## References

- [1] M. J. van Dort, P. H. Woerlee, and A. J. Walker, "A Simple Model for Quantisation Effects in Heavily-Doped Silicon MOSFETs at Inversion Conditions," *Solid-State Electronics*, vol. 37, no. 3, pp. 411–414, 1994.
- [2] S. A. Harelund *et al.*, "A Simple Model for Quantum Mechanical Effects in Hole Inversion Layers in Silicon PMOS Devices," *IEEE Transactions on Electron Devices*, vol. 44, no. 7, pp. 1172–1173, 1997.
- [3] M. G. Ancona and H. F. Tiersten, "Macroscopic physics of the silicon inversion layer," *Physical Review B*, vol. 35, no. 15, pp. 7959–7965, 1987.
- [4] M. G. Ancona and G. J. Iafrate, "Quantum correction to the equation of state of an electron gas in a semiconductor," *Physical Review B*, vol. 39, no. 13, pp. 9536–9540, 1989.
- [5] G. Paasch and H. Übensee, "A Modified Local Density Approximation: Electron Density in Inversion Layers," *Physica Status Solidi (b)*, vol. 113, no. 1, pp. 165–178, 1982.
- [6] G. Paasch and H. Übensee, "Carrier Density near the Semiconductor–Insulator Interface: Local Density Approximation for Non-Isotropic Effective Mass," *Physica Status Solidi (b)*, vol. 118, no. 1, pp. 255–266, 1983.
- [7] O. Penzin *et al.*, "Extended Quantum Correction Model Applied to Six-Band k·p Valence Bands Near Silicon/Oxide Interfaces," *IEEE Transactions on Electron Devices*, vol. 58, no. 6, pp. 1614–1619, 2011.
- [8] G. Paasch, T. Fiedler, and I. Bartoš, "Subband Energies at Semiconductor Heterojunctions: An Approximate Quantization Formula," *Physica Status Solidi (b)*, vol. 134, no. 2, pp. 825–835, 1986.
- [9] O. Penzin, G. Paasch, and L. Smith, "Nonparabolic Multivalley Quantum Correction Model for InGaAs Double-Gate Structures," *IEEE Transactions on Electron Devices*, vol. 60, no. 7, pp. 2246–2250, 2013.

# 15

## Mobility

---

*This chapter describes the mobility models implemented in Sentaurus Device.*

---

### Introduction to Mobility Models

Sentaurus Device uses a modular approach for the description of the carrier mobilities. In the simplest case, mobility is a function of the lattice temperature. The constant mobility model should be used only for undoped materials (see [Mobility due to Phonon Scattering on page 386](#)). For doped materials, the carriers scatter with the impurities. This leads to a degradation of the mobility. [Doping-Dependent Mobility Degradation on page 387](#) introduces the models that describe this effect.

Models that describe the effects of carrier–carrier scattering are presented in [Carrier–Carrier Scattering on page 396](#). The Philips unified mobility model is a well-calibrated model that accounts for both impurity and carrier–carrier scattering (see [Philips Unified Mobility Model on page 398](#)).

Models that describe the mobility degradation at interfaces, for example, the silicon–oxide interface in the channel region of a MOSFET, are introduced in [Mobility Degradation at Interfaces on page 403](#). These models account for the scattering with surface phonons and surface roughness.

Finally, models that describe mobility degradation in high electric fields are discussed in [High-Field Saturation Models on page 438](#). A flexible model for hydrodynamic simulations is described in [Energy-Dependent Mobility on page 878](#).

---

### Combining Mobility Models

Mobility models are selected in the `Physics` section as options to `Mobility`, `eMobility`, or `hMobility`:

```
Physics{ Mobility( <arguments> ) ... }
```

Specifications with `eMobility` apply to electrons; specifications with `hMobility` apply to holes; and specifications with `Mobility` apply to both carrier types.

## Chapter 15: Mobility

### Mobility due to Phonon Scattering

If more than one mobility model is activated for a carrier type, then the different mobility contributions ( $\mu_1, \mu_2, \dots$ ) for bulk, surface mobility, and thin layers are combined by Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_1} + \frac{1}{\mu_2} + \dots \quad (249)$$

If the high-field saturation model is activated, the final mobility is computed in two steps. First, the low field mobility  $\mu_{\text{low}}$  is determined according to [Equation 249](#). Second, the final mobility is computed from a (model-dependent) formula as a function of a driving force  $F_{\text{hfs}}$ :

$$\mu = f(\mu_{\text{low}}, F_{\text{hfs}}) \quad (250)$$

---

## Mobility due to Phonon Scattering

The constant mobility model [1] is active by default. It accounts only for phonon scattering and, therefore, it is dependent only on the lattice temperature:

$$\mu_{\text{const}} = \mu_L \left( \frac{T}{300 \text{K}} \right)^{-\zeta} \quad (251)$$

where  $\mu_L$  is the mobility due to bulk phonon scattering. The default values of  $\mu_L$  and the exponent  $\zeta$  are listed in [Table 47](#). The parameters of the constant mobility model are accessible in the `ConstantMobility` parameter set.

*Table 47 Constant mobility model: Default coefficients for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$\mu_L$	<code>mumax</code>	1417	470.5	$\text{cm}^2/\text{Vs}$
$\zeta$	<code>exponent</code>	2.5	2.2	1

In some special cases, it might be necessary to deactivate the default constant mobility. This can be accomplished by specifying the `-ConstantMobility` option to `Mobility`:

```
Physics { Mobility ( -ConstantMobility ... ) ... }
```

In [Equation 251](#), the exponent of the temperature ( $\zeta$ ) is a constant, which corresponds to the default behavior. For a more accurate treatment of mobility as a function of temperature (particularly, mobility at low temperatures), the exponent becomes a function of temperature itself. For more details, see [Low-Temperature Phonon Exponent Correction for Mobility Models on page 435](#) and [Constant Mobility Model on page 436](#).

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

---

## Doping-Dependent Mobility Degradation

In doped semiconductors, scattering of the carriers by charged impurity ions leads to degradation of the carrier mobility. Sentaurus Device supports several built-in models (see [Masetti Model on page 388](#), [Arora Model on page 389](#), and [University of Bologna Bulk Mobility Model on page 390](#)), one multistate configuration-dependent model (see [The pmi\\_msc\\_mobility Model on page 393](#)), and two types of PMI for doping-dependent mobility (see [PMIs for Bulk Mobility on page 394](#)).

The Philips unified mobility model is also available to account for doping-dependent scattering. This model accounts for other effects as well (such as electron–hole scattering and screening of impurities by carriers) (see [Philips Unified Mobility Model on page 398](#)).

For very-short channel devices, it has been found that an additional contribution to low-field mobility is sometimes needed to match measured device characteristics. For this purpose, a ballistic mobility model is provided (see [Low-Field Ballistic Mobility Models on page 394](#)) that can be specified in addition to one of the doping-dependent mobility models described in this section.

---

## Using Doping-Dependent Mobility

Mobility degradation due to impurity scattering is activated by specifying the `DopingDependence` option to `Mobility`. The different model choices are selected as options to `DopingDependence`:

```
Physics {
    Mobility(
        DopingDependence (
            [ Masetti | Arora | UniBo | PhuMob | PhuMob2 | 
              PMIModel(Name= "<msc-dependent-bulk-model>" ...) |
              BalMob([Lch=<float>]) |
              <doping-dependent-pmi-model>
            ]
        )
    )
}
```

If `DopingDependence` is specified without options, Sentaurus Device uses a material-dependent default. For example, in silicon, the default is the `Masetti` model; for GaAs, the default is the `Arora` model. The default model (`Masetti` or `Arora`) for each material can be specified by using the parameter `formula`, which is accessible in the `DopingDependence` parameter set in the parameter file:

```
DopingDependence: {
    formula= 1 , 1      # [1]
}
```

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

When `formula=1`, the Masetti model is the default. To use the Arora model as the default, specify `formula=2`.

The `UniBo` option selects the University of Bologna bulk mobility model. The options `PhuMob` and `PhuMob2` select the Philips unified mobility model and an alternative version of this model, respectively (see [Philips Unified Mobility Model on page 398](#)).

## Using More Than One Doping-Dependent Mobility Model

In most cases, only one doping-dependent mobility model should be specified to avoid double-counting mobility effects. However, Sentaurus Device allows more than one model to be used in a simulation. It can be useful, for example, to include additional degradation components that are created as PMI models into the bulk mobility calculation. When more than one model is specified as an option to `DopingDependence`, they are combined using Matthiessen's rule. For example, the specification:

```
Physics {
    Mobility(DopingDependence(Masetti pmi_model1 pmi_model2) ...)
    ...
}
```

calculates the total bulk mobility using:

$$\frac{1}{\mu_b} = \frac{1}{\mu_{\text{Masetti}}} + \frac{1}{\mu_{\text{pmi\_model1}}} + \frac{1}{\mu_{\text{pmi\_model2}}} \quad (252)$$

---

## Masetti Model

The default model used by Sentaurus Device to simulate doping-dependent mobility in silicon was proposed by Masetti *et al.* [2]:

$$\mu_{\text{dop}} = \mu_{\text{min1}} \exp \left( -\frac{P_c}{N_{A,0} + N_{D,0}} \right) + \frac{\mu_{\text{const}} - \mu_{\text{min2}}}{1 + ((N_{A,0} + N_{D,0})/C_r)^\alpha} - \frac{\mu_1}{1 + (C_s/(N_{A,0} + N_{D,0}))^\beta} \quad (253)$$

The reference mobilities  $\mu_{\text{min1}}$ ,  $\mu_{\text{min2}}$ , and  $\mu_1$ , the reference doping concentrations  $P_c$ ,  $C_r$ , and  $C_s$ , and the exponents  $\alpha$  and  $\beta$  are accessible in the parameter set `DopingDependence` in the parameter file. The corresponding values for silicon are given in [Table 48](#).

Table 48     Masetti model: Default coefficients

Symbol	Parameter name	Electrons	Holes	Unit
$\mu_{\text{min1}}$	<code>mumin1</code>	52.2	44.9	$\text{cm}^2/\text{Vs}$
$\mu_{\text{min2}}$	<code>mumin2</code>	52.2	0	$\text{cm}^2/\text{Vs}$

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

Table 48 Masetti model: Default coefficients (Continued)

Symbol	Parameter name	Electrons	Holes	Unit
$\mu_1$	mul	43.4	29.0	cm <sup>2</sup> /Vs
$P_c$	pc	0	$9.23 \times 10^{16}$	cm <sup>-3</sup>
$C_r$	cr	$9.68 \times 10^{16}$	$2.23 \times 10^{17}$	cm <sup>-3</sup>
$C_s$	cs	$3.43 \times 10^{20}$	$6.10 \times 10^{20}$	cm <sup>-3</sup>
$\alpha$	alpha	0.680	0.719	1
$\beta$	beta	2.0	2.0	1

The low-doping reference mobility  $\mu_{\text{const}}$  is determined by the constant mobility model (see [Mobility due to Phonon Scattering on page 386](#)).

## Arora Model

The Arora model [3] reads:

$$\mu_{\text{dop}} = \mu_{\text{min}} + \frac{\mu_d}{1 + ((N_{A,0} + N_{D,0})/N_0)^{A^*}} \quad (254)$$

with:

$$\mu_{\text{min}} = A_{\text{min}} \cdot \left( \frac{T}{300 \text{ K}} \right)^{\alpha_m} \quad ; \quad \mu_d = A_d \cdot \left( \frac{T}{300 \text{ K}} \right)^{\alpha_d} \quad (255)$$

and:

$$N_0 = A_N \cdot \left( \frac{T}{300 \text{ K}} \right)^{\alpha_N} \quad ; \quad A^* = A_a \cdot \left( \frac{T}{300 \text{ K}} \right)^{\alpha_a} \quad (256)$$

The parameters are accessible in the `DopingDependence` parameter set in the parameter file.

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

Table 49 Arora model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
$A_{\min}$	Ar_mumin	88	54.3	cm <sup>2</sup> /Vs
$\alpha_m$	Ar_alm	-0.57	-0.57	1
$A_d$	Ar_mud	1252	407	cm <sup>2</sup> /Vs
$\alpha_d$	Ar_ald	-2.33	-2.23	1
$A_N$	Ar_N0	$1.25 \times 10^{17}$	$2.35 \times 10^{17}$	cm <sup>-3</sup>
$\alpha_N$	Ar_alN	2.4	2.4	1
$A_a$	Ar_a	0.88	0.88	1
$\alpha_a$	Ar_ala	-0.146	-0.146	1

---

## University of Bologna Bulk Mobility Model

The University of Bologna bulk mobility model was developed for an extended temperature range between 25°C and 973°C. It should be used together with the University of Bologna inversion layer mobility model (see [University of Bologna Surface Mobility Model on page 416](#)). The model [4][5] is based on the Masetti approach with two major extensions. First, attractive and repulsive scattering are separately accounted for, therefore, leading to a function of both donor and acceptor concentrations. This automatically accounts for different mobility values for majority and minority carriers, and ensures continuity at the junctions as long as the impurity concentrations are continuous functions. Second, a suitable temperature dependence for most model parameters is introduced to predict correctly the temperature dependence of carrier mobility in a wider range of temperatures, with respect to other models. The temperature dependence of lattice mobility is reworked, with respect to the default temperature.

The model for lattice mobility is:

$$\mu_L(T) = \mu_{\max} \frac{\gamma + c \frac{T}{300 \text{ K}}}{\gamma - c \frac{T}{300 \text{ K}}} \quad (257)$$

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

where  $\mu_{\max}$  denotes the lattice mobility at room temperature, and  $c$  gives a correction to the lattice mobility at higher temperatures. The maximum mobility  $\mu_{\max}$  and the exponents  $\gamma$  and  $c$  are accessible in the `UniBoDopingDependence` parameter set in the parameter file.

The model for bulk mobility reads:

$$\mu_{\text{dop}}(T) = \mu_0(T) + \frac{\mu_L(T) - \mu_0(T)}{1 + \left[ \frac{N_{D,0}}{C_{r1}(T)} \right]^\alpha + \left[ \frac{N_{A,0}}{C_{r2}(T)} \right]^\beta} - \frac{\mu_1(N_{D,0}, N_{A,0}, T)}{1 + \left[ \frac{N_{D,0}}{C_{s1}(T)} + \frac{N_{A,0}}{C_{s2}(T)} \right]^2} \quad (258)$$

In turn,  $\mu_0$  and  $\mu_1$  are expressed as weighted averages of the corresponding limiting values for pure acceptor-doping and pure donor-doping densities:

$$\mu_0(T) = \frac{\mu_{0d}N_{D,0} + \mu_{0a}N_{A,0}}{N_{A,0} + N_{D,0}} \quad (259)$$

$$\mu_1(T) = \frac{\mu_{1d}N_{D,0} + \mu_{1a}N_{A,0}}{N_{A,0} + N_{D,0}} \quad (260)$$

The reference mobilities  $\mu_{0d}$ ,  $\mu_{0a}$ ,  $\mu_{1d}$ , and  $\mu_{1a}$ , and the reference doping concentrations  $C_{r1}$ ,  $C_{r2}$ ,  $C_{s1}$ , and  $C_{s2}$  are accessible in the `UniBoDopingDependence` parameter set in the parameter file.

[Table 50](#) lists the corresponding values of silicon for arsenic, phosphorus, and boron;  $T_n = T/300$  K.

The default parameters of Sentaurus Device for electrons are those for arsenic. The bulk mobility model was calibrated with experiments [5] in the temperature range from 300 K to 700 K. It is suitable for isothermal simulations at large temperatures or nonisothermal simulations.

*Table 50 Parameters of University of Bologna bulk mobility model*

Symbol	Parameter name	Electrons (As)	Electrons (P)	Holes (B)	Unit
$\mu_{\max}$	<code>mumax</code>	1441	1441	470.5	$\text{cm}^2/\text{Vs}$
$c$	<code>Exponent2</code>	-0.11	-0.11	0	1
$\gamma$	<code>Exponent</code>	2.45	2.45	2.16	1
$\mu_{0d}$	<code>mumin1</code>	$55.0T_n^{-\gamma_{0d}}$	$62.2T_n^{-\gamma_{0d}}$	$90.0T_n^{-\gamma_{0d}}$	$\text{cm}^2/\text{Vs}$
$\gamma_{0d}$	<code>mumin1_exp</code>	0.6	0.7	1.3	1
$\mu_{0a}$	<code>mumin2</code>	$132.0T_n^{-\gamma_{0a}}$	$132.0T_n^{-\gamma_{0a}}$	$44.0T_n^{-\gamma_{0a}}$	$\text{cm}^2/\text{Vs}$

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

*Table 50 Parameters of University of Bologna bulk mobility model (Continued)*

Symbol	Parameter name	Electrons (As)	Electrons (P)	Holes (B)	Unit
$\gamma_{0a}$	mumin2_exp	1.3	1.3	0.7	1
$\mu_{1d}$	mul	$42.4T_n^{-\gamma_{1d}}$	$48.6T_n^{-\gamma_{1d}}$	$28.2T_n^{-\gamma_{1d}}$	$\text{cm}^2/\text{Vs}$
$\gamma_{1d}$	mul1_exp	0.5	0.7	2.0	1
$\mu_{1a}$	mu2	$73.5T_n^{-\gamma_{1a}}$	$73.5T_n^{-\gamma_{1a}}$	$28.2T_n^{-\gamma_{1a}}$	$\text{cm}^2/\text{Vs}$
$\gamma_{1a}$	mu2_exp	1.25	1.25	0.8	1
$C_{r1}$	Cr	$8.9 \times 10^{16} T_n^{\gamma_{r1}}$	$8.5 \times 10^{16} T_n^{\gamma_{r1}}$	$1.3 \times 10^{18} T_n^{\gamma_{r1}}$	$\text{cm}^{-3}$
$\gamma_{r1}$	Cr_exp	3.65	3.65	2.2	1
$C_{r2}$	Cr2	$1.22 \times 10^{17} T_n^{\gamma_{r2}}$	$1.22 \times 10^{17} T_n^{\gamma_{r2}}$	$2.45 \times 10^{17} T_n^{\gamma_{r2}}$	$\text{cm}^{-3}$
$\gamma_{r2}$	Cr2_exp	2.65	2.65	3.1	1
$C_{s1}$	Cs	$2.9 \times 10^{20} T_n^{\gamma_{s1}}$	$4.0 \times 10^{20} T_n^{\gamma_{s1}}$	$1.1 \times 10^{18} T_n^{\gamma_{s1}}$	$\text{cm}^{-3}$
$\gamma_{s1}$	Cs_exp	0	0	6.2	1
$C_{s2}$	Cs2	$7.0 \times 10^{20}$	$7.0 \times 10^{20}$	$6.1 \times 10^{20}$	$\text{cm}^{-3}$
$\alpha$	alpha	0.68	0.68	0.77	1
$\beta$	beta	0.72	0.72	0.719	1

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

## The pmi\_msc\_mobility Model

The mobility model `pmi_msc_mobility` depends on the lattice temperature and the occupation probabilities of the states of a multistate configuration (MSC). The model averages the mobilities of all MSC states according to:

$$\mu = \sum_i \mu_i(T) s_i \quad (261)$$

where the sum is taken over all MSC states,  $\mu_i$  are the state mobilities, and  $s_i$  are the state occupation probabilities.

Each state mobility can be temperature dependent according to:

$$\mu_i = \begin{cases} \mu_{i, \text{ref}} & \text{if } T < T_{\text{ref}} \\ \frac{1}{T_g - T_{\text{ref}}} [(T_g - T)\mu_{i, \text{ref}} + (T - T_{\text{ref}})\mu_{i, g}] & \text{if } T_{\text{ref}} \leq T < T_g \\ \mu_{i, g} & \text{if } T_g \leq T \end{cases} \quad (262)$$

where:

- The index  $i$  refers to the state.
- $T_{\text{ref}}$  and  $T_g$  are the reference and glass temperatures (terminology is borrowed from and the model is used for phase transition dynamics).
- $\mu_{i, \text{ref}}$  and  $\mu_{i, g}$  are the (state-specific) mobilities for the corresponding temperatures.

The model is activated (here, for MSC "m0" and model string "e") by:

```
PMIModel ( Name="pmi_msc_mobility" MSConfig="m0" String="e" )
```

The given `String` is used to determine the names of parameters in the parameter file.

*Table 51 Global and model-string parameters of pmi\_msc\_mobility*

Name	Symbol	Default	Unit	Range	Description
plot	—	0	—	{0,1}	Plot parameter to screen
Tref	$T_{\text{ref}}$	300.	K	$\geq 0.$	Reference temperature
Tg	$T_g$	-1.	K	$\geq T_{\text{ref}}$ or equal -1.	Glass temperature
mu_ref	—	1.	cm/Vs	$> 0.$	Value at reference temperature
mu_g	—	1.	cm/Vs	$> 0.$	Value at glass temperature

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

The model uses a three-level hierarchy of sets of parameters to define the parameters for each state, namely, the global, the model-string, and the state parameters. The global parameters of the model are given in [Table 51](#). They serve as defaults for the model-string parameters, which are the global parameters with the prefix:

<model\_string>\_

where <model\_string> is the `String` parameter given in the command file. The model-string parameters, in turn, serve as defaults for the state parameters (see [Table 52](#)), which are prefixed by:

<model\_string>\_<state\_name>\_

where <state\_name> is the name of the MSC state.

*Table 52 State parameters of pmi\_msc\_mobility*

Name	Symbol	Default	Unit	Range	Description
mu_ref	$\mu_{i, \text{ref}}$	—	cm <sup>2</sup> /Vs	>0.	Value at reference temperature
mu_g	$\mu_{i, g}$	—	cm <sup>2</sup> /Vs	>0.	Value at glass temperature

---

## PMIs for Bulk Mobility

The PMIs for bulk mobility are described in [Doping-Dependent Mobility on page 1279](#) and [Multistate Configuration–Dependent Apparent Band-Edge Shift on page 1323](#).

---

## Low-Field Ballistic Mobility Models

You can use one of the following models to describe ballistic mobility.

### Simple Channel Length–Dependent Ballistic Mobility Model

This model is based on Shur [6] and is given by the following simple expression, where ballistic mobility is proportional to the channel length:

$$\mu_{\text{bal}} = k \cdot L_{\text{ch}} \quad (263)$$

Both  $k$  and  $L_{\text{ch}}$  are model parameters. In most cases, you specify a value for  $L_{\text{ch}}$  (in nm) that corresponds to the actual channel length for the device being simulated. You can adjust the parameter  $k$  to match measured device characteristics.

## Chapter 15: Mobility

### Doping-Dependent Mobility Degradation

*Table 53 Simple ballistic mobility model: Default coefficients for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$k$	$k$	20.0	20.0	$\text{cm}^2/\text{Vs nm}$
$L_{\text{ch}}$	$L_{\text{ch}}$	$10^7$	$10^7$	nm

## Injection Velocity–Dependent Ballistic Mobility Model

This model includes the finite injection velocity from the source and drain with the mobility concept [7] and is given by the following expression:

$$\mu_{\text{bal}} = f \cdot L_{\text{ch}} \quad (264)$$

Here,  $f$  is a function of injection velocity. The functional form of  $f$  is given as follows:

$$f = \frac{1}{V_{\text{ds}} + V_b} \frac{V_b}{V_{\text{ds,lin}}} \left[ \frac{L_{\text{ch}}}{L_{\text{ref}}} \right]^{\alpha} \frac{V_{\text{inj}}}{V_{\text{inj}}} \frac{1 - \exp -\frac{V_{\text{ds}}}{k_B T}}{1 + \exp -\frac{V_{\text{ds}}}{k_B T}} \quad (265)$$

where:

- $\alpha$ ,  $L_{\text{ch}}$ ,  $L_{\text{ref}}$ ,  $V_b$ ,  $V_{\text{ds}}$ ,  $V_{\text{ds,lin}}$ , and  $V_{\text{inj}}$  are model parameters (see Table 54).
- $T$  is the lattice temperature.
- $k_B$  is the Boltzmann constant.

*Table 54 Injection velocity ballistic mobility model: Default coefficients for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$\alpha$	alpha	0.0	0.0	1
$L_{\text{ch}}$	$L_{\text{ch}}$	$10^7$	$10^7$	nm
$L_{\text{ref}}$	$L_{\text{ref}}$	100	100	nm
$V_b$	$v_{\text{bbal}}$	0.0	0.0	V
$V_{\text{ds}}$	$v_{\text{dsbal}}$	0.75	0.75	V

## Chapter 15: Mobility

### Carrier–Carrier Scattering

**Table 54** Injection velocity ballistic mobility model: Default coefficients for silicon (Continued)

Symbol	Parameter name	Electrons	Holes	Unit
$V_{ds,lin}$	vdslin	0.05	0.05	V
$V_{inj}$	vinj	3.0e7	3.0e7	cm/s

To use a low-field ballistic mobility model, specify `BalMob` as an argument to `DopingDependence`. For example:

```
Physics {
    Mobility (
        DopingDependence ( PhuMob BalMob(Lch = 5.0) )
    )
}
```

In this case,  $\mu_{bal}$  is combined with the mobility from `PhuMob` by using Matthiessen's rule. If you do not specify `Lch`, then the default value of  $10^7$  nm is used, which effectively deactivates the model.

By default, the simple channel length–dependent ballistic mobility model is activated by specifying `BalMob`. The injection velocity–dependent ballistic mobility model is invoked by setting a positive nonzero value for the  $V_b$  parameter in the parameter file.

## Carrier–Carrier Scattering

Two models are supported for the description of carrier–carrier scattering. One model is based on Choo [8] and Fletcher [9], and uses the Conwell–Weisskopf theory. As an alternative to the Conwell–Weisskopf model, Sentaurus Device supports a model based on the Brooks–Herring screening theory [10]. The carrier–carrier contribution to the overall mobility degradation is captured in the mobility term  $\mu_{eh}$ . This is combined with the mobility contributions from other degradation models ( $\mu_{other}$ ) according to Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{other}} + \frac{1}{\mu_{eh}} \quad (266)$$

## Using Carrier–Carrier Scattering

Carrier–carrier scattering models are activated by specifying the `CarrierCarrierScattering` option to `Mobility`.

## Chapter 15: Mobility

### Carrier–Carrier Scattering

Either of the two different models are selected by an additional flag:

```
Physics { Mobility (
    CarrierCarrierScattering( [ ConwellWeisskopf | BrooksHerring ] )
    ... ) ... }
```

The Conwell–Weisskopf model is the default in Sentaurus Device for carrier–carrier scattering and is activated when `CarrierCarrierScattering` is specified without an option.

---

## Conwell–Weisskopf Model

$$\mu_{\text{eh}} = \frac{D(T/300\text{ K})^{3/2}}{\sqrt{np}} \left[ \ln \left( 1 + F \frac{T}{300\text{ K}}^2 (pn)^{-1/3} \right) \right]^{-1} \quad (267)$$

The parameters  $D$  and  $F$  are accessible in the parameter file. The default values appropriate for silicon are given in [Table 55](#).

---

## Brooks–Herring Model

$$\mu_{\text{eh}} = \frac{c_1(T/300\text{ K})^{3/2}}{\sqrt{np}} \frac{1}{\phi(\eta_0)} \quad (268)$$

where  $\phi(\eta_0) = \ln(1 + \eta_0) - \eta_0/(1 + \eta_0)$  and:

$$\eta_0(T) = \frac{c_2}{N_C F_{-1/2}(n/N_C) + N_V F_{-1/2}(p/N_V)} \frac{T}{300\text{ K}}^2 \quad (269)$$

[Table 56](#) lists the silicon default values for  $c_1$  and  $c_2$ .

---

## Physical Model Parameters

Parameters for the carrier–carrier scattering models are accessible in the parameter set `CarrierCarrierScattering`.

*Table 55 Conwell–Weisskopf model: Default parameters*

Symbol	Parameter name	Value	Unit
$D$	<code>D</code>	$1.04 \times 10^{21}$	$\text{cm}^{-1}\text{V}^{-1}\text{s}^{-1}$
$F$	<code>F</code>	$7.452 \times 10^{13}$	$\text{cm}^{-2}$

## Chapter 15: Mobility

### Philips Unified Mobility Model

Table 56 Brooks-Herring model: Default parameters

Symbol	Parameter name	Value	Unit
$c_1$	c1	$1.56 \times 10^{21}$	$\text{cm}^{-1}\text{V}^{-1}\text{s}^{-1}$
$c_2$	c2	$7.63 \times 10^{19}$	$\text{cm}^{-3}$

---

## Philips Unified Mobility Model

The Philips unified mobility model, proposed by Klaassen [11], unifies the description of majority and minority carrier bulk mobilities. In addition to describing the temperature dependence of the mobility, the model takes into account electron–hole scattering, screening of ionized impurities by charge carriers, and clustering of impurities.

The Philips unified mobility model is well calibrated. Though it was initially used primarily for bipolar devices, it is widely used for MOS devices.

---

## Using the Philips Unified Mobility Model

There are different methods for activating the Philips unified mobility model.

As described in [Using Doping-Dependent Mobility on page 387](#), you can activate the model by specifying `PhuMob` as an option to `DopingDependence`. This method is required if you want to combine `PhuMob` with an additional doping-dependent model (for example, a doping-dependent PMI model to account for other degradation effects).

Alternatively, you can activate the Philips unified mobility model by specifying the `PhuMob` option to `Mobility` directly:

```
Physics{ Mobility ( PhuMob ... ) ... }
```

Specifying `PhuMob` in this way causes Sentaurus Device to completely ignore any specification made with `DopingDependence`, if present, to avoid accidental double-counting of mobility effects.

The `PhuMob` model also can be activated with an additional option:

```
Physics{ Mobility ( PhuMob[(Arsenic | Phosphorus)] ... ) ... }
```

The option `Arsenic` or `Phosphorus` specifies which parameters (see [Table 58 on page 402](#)) are used. These parameters reflect the different electron mobility degradation that is observed in the presence of these donor species.

## Chapter 15: Mobility

Philips Unified Mobility Model

### Note:

The Philips unified mobility model describes mobility degradation due to both impurity scattering and carrier–carrier scattering mechanisms. Therefore, the keyword `PhuMob` must not be combined with the keyword `DopingDependence` or `CarrierCarrierScattering`. If a combination of these keywords is specified, Sentaurus Device uses only the Philips unified mobility model.

---

## Using an Alternative Philips Unified Mobility Model

Sentaurus Device provides an extended PMI reimplementation of the Philips unified mobility model (see [Doping-Dependent Mobility on page 1279](#)). To use it, specify `PhuMob2` as the option for `DopingDependence`:

```
Physics { Mobility (DopingDependence(PhuMob2) . . . ) . . . }
```

The reimplementation uses the `PhuMob2` parameter set. The parameter `FACT_G` has a default value of 1. It modifies the function  $G(P_i)$  shown in [Equation 282](#) by a factor  $G(P_i) = \text{FACT\_G}$  (right-hand side in [Equation 282](#)). Otherwise, the reimplementation fully agrees with the model described below.

---

## Description of the Philips Unified Mobility Model

According to the Philips unified mobility model, there are two contributions to carrier mobilities. The first,  $\mu_{i, L}$ , represents phonon (lattice) scattering and the second,  $\mu_{i, DAeh}$ , accounts for all other bulk scattering mechanisms (due to free carriers, and ionized donors and acceptors).

These partial mobilities are combined to give the bulk mobility  $\mu_{i, b}$  for each carrier according to Matthiessen's rule:

$$\frac{1}{\mu_{i, b}} = \frac{1}{\mu_{i, L}} + \frac{1}{\mu_{i, DAeh}} \quad (270)$$

In [Equation 270](#) and all of the following model equations, the index  $i$  takes the value 'e' for electrons and 'h' for holes. The first contribution due to lattice scattering takes the form:

$$\mu_{i, L} = \mu_{i, \max} \left[ \frac{T}{300 \text{ K}} \right]^{-\theta_i} \quad (271)$$

The second contribution has the form:

$$\mu_{i, DAeh} = \mu_{i, N} \left[ \frac{N_{i, sc}}{N_{i, sc, eff}} \right]^{\alpha_i} + \mu_{i, c} \left[ \frac{n + p}{N_{i, sc, eff}} \right] \quad (272)$$

## Chapter 15: Mobility

### Philips Unified Mobility Model

with:

$$\mu_{i,N} = \frac{\mu_{i,\max}^2}{\mu_{i,\max} - \mu_{i,\min}} \left[ \frac{T}{300 \text{ K}} \right]^{3\alpha_i - 1.5} \quad (273)$$

$$\mu_{i,c} = \frac{\mu_{i,\max} \mu_{i,\min}}{\mu_{i,\max} - \mu_{i,\min}} \left[ \frac{300 \text{ K}}{T} \right]^{0.5} \quad (274)$$

for electrons:

$$N_{e,sc} = N_D^* + N_A^* + p \quad (275)$$

$$N_{e,sc,eff} = N_D^* + G(P_e)N_A^* + f_e \frac{p}{F(P_e)} \quad (276)$$

and for holes:

$$N_{h,sc} = N_A^* + N_D^* + n \quad (277)$$

$$N_{h,sc,eff} = N_A^* + G(P_h)N_D^* + f_h \frac{n}{F(P_h)} \quad (278)$$

The effects of clustering of donors ( $N_D^*$ ) and acceptors ( $N_A^*$ ) at ultrahigh concentrations are described by ‘clustering’ functions  $Z_D$  and  $Z_A$ , which are defined as:

$$N_D^* = N_{D,0}Z_D = N_{D,0} \left[ 1 + \frac{N_{D,0}^2}{c_D N_{D,0}^2 + N_{D,ref}^2} \right] \quad (279)$$

$$N_A^* = N_{A,0}Z_A = N_{A,0} \left[ 1 + \frac{N_{A,0}^2}{c_A N_{A,0}^2 + N_{A,ref}^2} \right] \quad (280)$$

The analytic functions  $G(P_i)$  and  $F(P_i)$  in Equation 276 and Equation 278 describe minority impurity and electron–hole scattering. They are given by:

$$F(P_i) = \frac{0.7643 P_i^{0.6478} + 2.2999 + 6.5502(m^*_i/m^*_j)}{P_i^{0.6478} + 2.3670 - 0.8552(m^*_i/m^*_j)} \quad (281)$$

and:

$$G(P_i) = 1 - a_g \left[ b_g + P_i \left[ \frac{m_0}{m^*_i 300 \text{ K}} \right]^{\alpha_g} \right]^{-\beta_g} + \frac{c_g}{c_g} \left[ P_i \left[ \frac{m^*_i 300 \text{ K}}{m_0 T} \right]^{\alpha'_g} \right]^{-\gamma_g} \quad (282)$$

where  $m^*_i$  denotes a fit parameter for carrier  $i$  (which is related to the effective carrier mass) and  $m^*_j$  denotes a fit parameter for the other carrier.

## Screening Parameter

The screening parameter  $P_i$  is given by a weighted harmonic mean of the Brooks–Herring approach and Conwell–Weisskopf approach:

$$P_i = \left[ \frac{f_{\text{CW}}}{3.97 \times 10^{13} \text{ cm}^{-2} N_{i,\text{sc}}^{-2/3}} + f_{\text{BH}} \frac{(n+p)}{1.36 \times 10^{20} \text{ cm}^{-3} m^*_i} \right]^{-1} \frac{T}{300 \text{ K}}^2 \quad (283)$$

The evaluation of  $G(P_i)$  depends on the value of the screening parameter  $P_i$ . For values of  $P_i < P_{i,\text{min}}$ ,  $G(P_{i,\text{min}})$  is used instead of  $G(P_i)$ , where  $P_{i,\text{min}}$  is the value at which  $G(P_i)$  reaches its minimum. If  $G(P_{i,\text{min}})$  is negative, which might happen in the model for the lattice temperature  $T$  below 50 K, then  $|G(P_{i,\text{min}})|$  is used for  $G(P_i) < |G(P_{i,\text{min}})|$ .

## Philips Model Parameters

[Table 57](#) lists the built-in values for the parameters  $a_g$ ,  $b_g$ ,  $c_g$ ,  $\alpha_g$ ,  $\alpha'_g$ ,  $\beta_g$ , and  $\gamma_g$  in [Equation 282](#).

*Table 57 Philips unified mobility model parameters*

Symbol	Value	Unit
$a_g$	0.89233	1
$b_g$	0.41372	1
$c_g$	0.005978	1
$\alpha_g$	0.28227	1
$\alpha'_g$	0.72169	1
$\beta_g$	0.19778	1
$\gamma_g$	1.80618	1

Other parameters for the Philips unified mobility model are accessible in the parameter set PhuMob.

[Table 58](#) and [Table 59](#) list the silicon defaults for other parameters. Sentaurus Device supports different parameters for electron mobility, which are optimized for situations where

## Chapter 15: Mobility

### Philips Unified Mobility Model

the dominant donor species in the silicon is either arsenic or phosphorus. The arsenic parameters are used by default.

*Table 58 Philips unified mobility model: Electron and hole parameters (silicon)*

Symbol	Parameter name	Electrons (arsenic)	Electrons (phosphorus)	Holes (boron)	Unit
$\mu_{\max}$	mumax_*	1417	1414	470.5	$\text{cm}^2/\text{Vs}$
$\mu_{\min}$	mumin_*	52.2	68.5	44.9	$\text{cm}^2/\text{Vs}$
$\theta$	theta_*	2.285	2.285	2.247	1
$N_{\{e,h\},\text{ref}}$	n_ref_*	$9.68 \times 10^{16}$	$9.2 \times 10^{16}$	$2.23 \times 10^{17}$	$\text{cm}^{-3}$
$\alpha$	alpha_*	0.68	0.711	0.719	1

The original Philips unified mobility model uses four fit parameters: the weight factors  $f_{\text{CW}}$  and  $f_{\text{BH}}$ , and the ‘effective masses’  $m_e^*$  and  $m_h^*$ . The optimal parameter set, determined by accurate fitting to experimental data [11] is shown in [Table 59](#). The Philips unified mobility model can be slightly modified by setting parameters  $f_e = 0$  and  $f_h = 0$  as required for the Lucent mobility model (see [Lucent Model on page 446](#)).

*Table 59 Philips unified mobility model: Other fitting parameters (silicon)*

Symbol	Parameter name	Value	Unit
$m_e^*/m_0$	me_over_m0	1	1
$m_h^*/m_0$	mh_over_m0	1.258	1
$f_{\text{CW}}$	f_CW	2.459	1
$f_{\text{BH}}$	f_BH	3.828	1
$f_e$	f_e	1.0	1
$f_h$	f_h	1.0	1

In [Equation 271](#), the exponent of the temperature ( $\theta$ ) is a constant, which corresponds to the default behavior. For a more accurate treatment of mobility as a function of temperature

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

(particularly, mobility at low temperatures), the exponent becomes a function of temperature itself. For more details, see [Low-Temperature Phonon Exponent Correction for Mobility Models on page 435](#) and [Philips Unified Mobility Model on page 436](#).

---

## Mobility Degradation at Interfaces

In the channel region of a MOSFET, the high transverse electric field forces carriers to interact strongly with the semiconductor–insulator interface. Carriers are subjected to scattering by acoustic surface phonons and surface roughness. The models in this section describe mobility degradation caused by these effects.

---

### Using Mobility Degradation at Interfaces

To activate mobility degradation at interfaces, select a method to compute the transverse field  $F_{\perp}$  (see [Computing Transverse Field on page 426](#)).

To select the calculation of field perpendicular to the semiconductor–insulator interface, specify the `Enormal` option to `Mobility`:

```
Physics { Mobility ( Enormal ... ) ... }
```

Alternatively, to select calculation of  $F_{\perp}$  perpendicular to current flow, specify:

```
Physics { Mobility ( ToCurrentEnormal ... ) ... }
```

To select a mobility degradation model, specify an option to `Enormal` or `ToCurrentEnormal`. Valid options are `Lombardi`, `IALMob`, `UniBo`, or a PMI model provided by users. In addition, one or more mobility degradation components can be specified, including `Coulomb2D`, `RCS`, `RPS`, `NegInterfaceCharge`, and `PosInterfaceCharge`. For example:

```
Physics { Mobility ( Enormal(UniBo) ... ) ... }
```

selects the University of Bologna model (see [University of Bologna Surface Mobility Model on page 416](#)). The default model is `Lombardi` (see [Enhanced Lombardi Model on page 404](#)).

**Note:**

The mobility degradation models discussed in this section are very sensitive to mesh spacing. It is recommended that you reduce the vertical mesh spacing to 0.1 nm in the silicon at the oxide interface underneath the gate. For the extensions of the `Lombardi` model (see [Equation 288](#)), even smaller spacing of 0.05 nm is appropriate. This fine spacing is required only in the two uppermost rows of mesh and can be increased progressively moving away from the interface.

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

You can use more than one interface degradation mobility model in a simulation. For example, it can be useful to include additional degradation components that are created as PMI models into the `Enormal` mobility calculation. When more than one model is specified as an option to `Enormal`, they are combined using Matthiessen's rule. For example, the specification:

```
Physics { Mobility (Enormal(Lombardi pmi_model1 pmi_model2) ...) ...}
```

calculates the total `Enormal` mobility using:

$$\frac{1}{\mu_{\text{Enormal}}} = \frac{1}{\mu_{\text{Lombardi}}} + \frac{1}{\mu_{\text{pmi\_model1}}} + \frac{1}{\mu_{\text{pmi\_model2}}} \quad (284)$$

#### Note:

By default, an absolute value of the transverse electric field is used in all interface degradation mobility models (see [Computing Transverse Field on page 426](#)). To define mobility degradation only for carriers that are pulled to the interface by the electric field, specify the option `MobEnormalSignDependence` in the `Math` section.

---

## Enhanced Lombardi Model

The surface contribution due to acoustic phonon scattering has the form:

$$\mu_{\text{ac}} = \frac{B}{F_{\perp}} + \frac{C((N_{A,0} + N_{D,0} + N_2)/N_0)^{\lambda}}{F_{\perp}^{1/3} (T/300\text{ K})^k} \quad (285)$$

and the contribution attributed to surface roughness scattering is given by:

$$\mu_{\text{sr}} = \frac{(F_{\perp}/F_{\text{ref}})^{A^*}}{\delta} + \frac{F_{\perp}^3}{\eta}^{-1} \quad (286)$$

These surface contributions to the mobility ( $\mu_{\text{ac}}$  and  $\mu_{\text{sr}}$ ) are then combined with the bulk mobility  $\mu_b$  according to Matthiessen's rule (see [Mobility due to Phonon Scattering on page 386](#) and [Doping-Dependent Mobility Degradation on page 387](#)):

$$\frac{1}{\mu} = \frac{1}{\mu_b} + \frac{D}{\mu_{\text{ac}}} + \frac{D}{\mu_{\text{sr}}} \quad (287)$$

The reference field  $F_{\text{ref}} = 1 \text{ V/cm}$  ensures a unitless numerator in [Equation 286](#).  $F_{\perp}$  is the transverse electric field normal to the semiconductor–insulator interface, see [Computing Transverse Field on page 426](#).  $D = \exp(-x/l_{\text{crit}})$  (where  $x$  is the distance from the interface and  $l_{\text{crit}}$  a fit parameter) is a damping that switches off the inversion layer terms far away from the interface. All other parameters are accessible in the parameter file.

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

In the Lombardi model [1], the exponent  $A^*$  in [Equation 286](#) is equal to 2. According to another study [12], an improved fit to measured data is achieved if  $A^*$  is given by:

$$A^* = A + \frac{(\alpha_{\perp,n}n + \alpha_{\perp,p}p)N_{\text{ref}}^v}{(N_{A,0} + N_{D,0} + N_1)^v} \quad (288)$$

where  $n$  and  $p$  denote the electron and hole concentrations, respectively. For electron mobility,  $\alpha_{\perp,n} = \alpha_{\perp}$  and  $\alpha_{\perp,p} = \alpha_{\perp}a_{\text{other}}$ ; for hole mobility,  $\alpha_{\perp,n} = \alpha_{\perp}a_{\text{other}}$  and  $\alpha_{\perp,p} = \alpha_{\perp}$ .

The reference doping concentration  $N_{\text{ref}} = 1 \text{ cm}^{-3}$  cancels the unit of the term raised to the power  $v$  in the denominator of [Equation 288](#). The Lombardi model parameters are accessible in the parameter set EnormalDependence.

[Table 60](#) lists the respective default parameters that are appropriate for silicon. The parameters  $B$ ,  $C$ ,  $N_0$ , and  $\lambda$  were fitted at SGS Thomson and are not in the literature [1].

*Table 60 Lombardi model: Default coefficients for silicon*

Symbol	Parameter	Electrons	Holes	Unit
$B$	B	$4.75 \times 10^7$	$9.925 \times 10^6$	cm/s
$C$	C	$5.80 \times 10^2$	$2.947 \times 10^3$	$\text{cm}^{5/3}\text{V}^{-2/3}\text{s}^{-1}$
$N_0$	N0	1	1	$\text{cm}^{-3}$
$N_2$	N2	1	1	$\text{cm}^{-3}$
$\lambda$	lambda	0.1250	0.0317	1
$k$	k	1	1	1
$\delta$	delta	$5.82 \times 10^{14}$	$2.0546 \times 10^{14}$	$\text{cm}^2/\text{Vs}$
$A$	A	2	2	1
$\alpha_{\perp}$	alpha	0	0	$\text{cm}^3$
$N_1$	N1	1	1	$\text{cm}^{-3}$
$v$	nu	1	1	1
$\eta$	eta	$5.82 \times 10^{30}$	$2.0546 \times 10^{30}$	$\text{V}^2\text{cm}^{-1}\text{s}^{-1}$
$a_{\text{other}}$	aother	0	0	1
$l_{\text{crit}}$	l_crit	$1 \times 10^{-6}$	$1 \times 10^{-6}$	cm
$a_{\text{ac}}$	a_ac	1.0	1.0	1
$a_{\text{sr}}$	a_sr	1.0	1.0	1

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

The modifications of the Lombardi model suggested in [12] can be activated by setting  $\alpha_{\perp}$  to a nonzero value.

**Table 61** lists a consistent set of parameters for the modified model.

**Table 61** Lombardi model: Lucent coefficients for silicon

Symbol	Parameter	Electrons	Holes	Unit
B	B	$3.61 \times 10^7$	$1.51 \times 10^7$	cm/s
C	C	$1.70 \times 10^4$	$4.18 \times 10^3$	$\text{cm}^{5/3} \text{V}^{-2/3} \text{s}^{-1}$
$N_0$	N0	1	1	$\text{cm}^{-3}$
$N_2$	N2	1	1	$\text{cm}^{-3}$
$\lambda$	lambda	0.0233	0.0119	1
k	k	1.7	0.9	1
$\delta$	delta	$3.58 \times 10^{18}$	$4.10 \times 10^{15}$	$\text{cm}^2/\text{Vs}$
A	A	2.58	2.18	1
$\alpha_{\perp}$	alpha	$6.85 \times 10^{-21}$	$7.82 \times 10^{-21}$	$\text{cm}^3$
$N_1$	N1	1	1	$\text{cm}^{-3}$
v	nu	0.0767	0.123	1
$\eta$	eta	$1 \times 10^{50}$	$1 \times 10^{50}$	$\text{V}^2 \text{cm}^{-1} \text{s}^{-1}$
$a_{\text{other}}$	aother	1	1	1
$l_{\text{crit}}$	l_crit	1	1	cm
$a_{\text{ac}}$	a_ac	1.0	1.0	1
$a_{\text{sr}}$	a_sr	1.0	1.0	1

## Stress Factors for Mobility Components

If isotropic stress-dependent mobility enhancement factors are applied to mobility components (see [Factor Models Applied to Mobility Components on page 995](#)), the Lombardi mobility components become:

$$\begin{aligned}\mu'_{\text{ac}} &= \gamma_{\text{ac}} \mu_{\text{ac}} \quad , \quad \gamma_{\text{ac}} = 1 + a_{\text{ac}}(\gamma - 1) \\ \mu'_{\text{sr}} &= \gamma_{\text{sr}} \mu_{\text{sr}} \quad , \quad \gamma_{\text{sr}} = 1 + a_{\text{sr}}(\gamma - 1)\end{aligned}\tag{289}$$

where  $a_{\text{ac}}$  and  $a_{\text{sr}}$  are stress scaling parameters that can be specified in the parameter set `EnormalDependence`, and  $\gamma$  is an isotropic enhancement factor calculated from a stress-dependent Factor model (see [Isotropic Factor Models on page 988](#)).

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

In [Equation 285](#), the exponent of the temperature ( $k$ ) is a constant, which corresponds to the default behavior. For a more accurate treatment of mobility as a function of temperature (particularly, mobility at low temperatures), the exponent becomes a function of temperature itself. For more details, see [Low-Temperature Phonon Exponent Correction for Mobility Models on page 435](#) and [Enhanced Lombardi Mobility Model on page 437](#).

## Named Parameter Sets for the Lombardi Model

The `EnormalDependence` parameter set can be named. For example, in the parameter file, you can write:

```
EnormalDependence "myset" { ... }
```

to declare a parameter set with the name `myset`. To use a named parameter set, specify its name with `ParameterSetName` as an option to `Lombardi` in the command file.

For example:

```
Mobility (
    Enormal( Lombardi( ParameterSetName = "myset" ... ) ... )
)
```

By default, the unnamed parameter set is used.

## Auto-Orientation for the Lombardi Model

The `Lombardi` model supports the auto-orientation framework (see [Auto-Orientation Framework on page 86](#)) that switches between different-named parameter sets based on the orientation of the nearest interface. This can be activated by specifying the `Lombardi` command file parameter `AutoOrientation`:

```
Mobility (
    Enormal( Lombardi( AutoOrientation ... ) ... )
)
```

---

## Inversion and Accumulation Layer Mobility Model

The inversion and accumulation layer mobility (`IALMob`) model is based on the *unified* portion of the model described in [\[13\]](#). This model includes doping and transverse-field dependencies and is similar to the Lucent (Darwish) model [\[12\]](#), but it contains additional terms that account for *two-dimensional* Coulomb impurity scattering. The implementation of the `IALMob` model is based on modified versions of the [Philips Unified Mobility Model on page 398](#) and the [Enhanced Lombardi Model on page 404](#), but it is completely self-contained and all parameters associated with this model are specified independently of those models. A complete description is given here.

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

The following expressions apply to both electron mobility and hole mobility when the following substitutions are made:

$$\begin{aligned} \text{Electron Mobility: } c &= n, c_{\text{other}} = p, N_{\text{inv}} = N_{A,0}, N_{\text{acc}} = N_{D,0}, P = P_e, m^* = m_e^*, m_{\text{other}}^* = m_h^* \\ \text{Hole Mobility: } c &= p, c_{\text{other}} = n, N_{\text{inv}} = N_{D,0}, N_{\text{acc}} = N_{A,0}, P = P_h, m^* = m_h^*, m_{\text{other}}^* = m_e^* \end{aligned} \quad (290)$$

The model has contributions from Coulomb impurity scattering, phonon scattering, and surface roughness scattering:

$$\frac{1}{\mu} = \frac{1}{\mu_C} + \frac{1}{\mu_{\text{ph}}} + \frac{D}{\mu_{\text{sr}}} \quad (291)$$

where  $D = \exp(-x/l_{\text{crit}})$  ( $x$  is the distance from the interface).

## Coulomb Scattering

The Coulomb impurity scattering term has 2D and 3D contributions. The 2D contribution is primarily due to effects occurring near the interface and the 3D contribution is primarily due to effects occurring in the bulk. These contributions are combined using a field-dependent function  $f(F_{\perp}, t, T)$  and a distance factor  $D_C$ :

$$\mu_C = \mu_{C,3D}(1 - D_C) + D_C[f(F_{\perp}, t, T)\mu_{C,3D} + (1 - f(F_{\perp}, t, T))\mu_{C,2D}] \quad (292)$$

where:

$$f(F_{\perp}, t, T) = \frac{1}{1 + \exp \frac{SF_{\perp}^{2/3}}{T} + \frac{S_t}{(t + t_0)^2 T} - p} \quad (293)$$

and  $D_C = \exp(-x/l_{\text{crit,C}})$ . In [Equation 293](#),  $t$  is the local layer thickness (see [Thickness Extraction on page 382](#)), which is calculated automatically when using the [ThinLayer](#) mobility model (see [Thin-Layer Mobility Model on page 427](#)) or if the [LayerThickness](#) command in the [Physics](#) section is used to calculate layer thickness.

Two-dimensional Coulomb scattering has both inversion and accumulation layer contributions, and also includes a dependency on the local layer thickness:

$$\frac{1}{\mu_{C,2D}} = \text{erf} \frac{t + t_1}{t_{\text{Coulomb}}} \frac{1}{\mu_{C,2D,\text{inv}}} + \frac{1}{\mu_{C,2D,\text{acc}}} \quad (294)$$

where:

$$\mu_{C,2D,\text{inv}} = \left[ \frac{D_{1,\text{inv}}(T/300\text{K})^{\alpha_{1,\text{inv}}} (c/N_{\text{sc,ref}})^{v_{0,\text{inv}}}}{(N_{\text{inv}}/N_{\text{dop,ref}})^{v_{1,\text{inv}}}} + \frac{D_{2,\text{inv}}(T/300\text{K})^{\alpha_{2,\text{inv}}}}{(N_{\text{inv}}/N_{\text{dop,ref}})^{v_{2,\text{inv}}}} \right]^{1/2} \quad (295)$$

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

$$\mu_{C,2D,acc} = \left[ \frac{D_{1,acc}(T/300K)^{\alpha_{1,acc}}(c/N_{sc,ref})^{v_{0,acc}}}{(N_{acc}/N_{dop,ref})^{v_{1,acc}}} + \frac{D_{2,acc}(T/300K)^{\alpha_{2,acc}}}{(N_{acc}/N_{dop,ref})^{v_{2,acc}}} \right]^{1/2} G(P) \quad (296)$$

The 3D Coulomb scattering part of [Equation 292](#) is taken from the Philips unified mobility model (`PhuMob`) and is given by:

$$\mu_{C,3D} = \mu_N \frac{N_{sc}}{N_{sc,eff}} \frac{N_{ref}}{N_{sc}}^{\alpha} + \mu_c \cdot \frac{c + c_{other}}{N_{sc,eff}} \quad (297)$$

where:

$$\mu_N = \frac{\mu_{max}^2}{\mu_{max} - \mu_{min}} \frac{T}{300K}^{3\alpha - 1.5} \quad (298)$$

$$\mu_c = \frac{\mu_{max}\mu_{min}}{\mu_{max} - \mu_{min}} \frac{300K}{T}^{1/2} \quad (299)$$

$$N_{sc} = N_D^* + N_A^* + c_{other} \quad (300)$$

$$N_{sc,eff} = N_{acc}^* + G(P)N_{inv}^* + \frac{c_{other}}{F(P)} \quad (301)$$

In the previous expressions, quantities marked with an asterisk (\*) indicate that the clustering formulas from the Philips unified mobility model are invoked:

$$N_D^* = N_{D,0} \left[ 1 + \frac{N_{D,0}^2}{c_D N_{D,0}^2 + N_{D,ref}^2} \right] \quad (302)$$

$$N_A^* = N_{A,0} \left[ 1 + \frac{N_{A,0}^2}{c_A N_{A,0}^2 + N_{A,ref}^2} \right] \quad (303)$$

Optionally, the clustering formulas can be used for all occurrences of  $N_A$  and  $N_D$  in the model by specifying the `IALMob` command file parameter `ClusteringEverywhere`.

The `AsPhPhuMob` option activates linear interpolation between the arsenic and phosphorus `PhuMob` parameter sets with respect to the arsenic and phosphorus concentrations.

The functions  $F(P)$  and  $G(P)$  describe electron–hole and minority impurity scattering, respectively, and  $P$  is a screening parameter:

$$F(P) = \frac{0.7643P^{0.6478} + 2.2999 + 6.5502(m^*/m^*_{other})}{P^{0.6478} + 2.3670 - 0.8552(m^*/m^*_{other})} \quad (304)$$

$$G(P) = 1 - \frac{0.89233}{\left[ 0.41372 + P \frac{m_0}{m} \frac{T}{300K} \right]^{0.28227}}^{0.19778} + \frac{0.005978}{\left[ P \frac{m}{m_0} \frac{300K}{T} \right]^{0.72169}}^{1.80618} \quad (305)$$

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

$$P = \left[ \frac{2.459}{3.97 \times 10^{13} \text{ cm}^{-2} N_{sc}^{2/3}} + \frac{3.828(c + c_{\text{other}})(m_0/m^*)}{1.36 \times 10^{20} \text{ cm}^{-3}} \right]^{-1} \frac{T}{300 \text{ K}}^2 \quad (306)$$

By default, the full PhuMob model described by the previous expressions is used (`FullPhuMob`). However, the ‘other’ carrier in these expressions can be ignored ( $c_{\text{other}} = 0$ ) by specifying `-FullPhuMob` in the command file. This specification also ignores the standard PhuMob modification of the  $G(P)$  function, which invokes  $G = G(P_{\min})$  for  $P < P_{\min}$ , where  $P_{\min}$  is the value where  $G(P)$  reaches its minimum. If  $G(P_{\min})$  is negative, which might happen in the model for the lattice temperature  $T$  below 50 K, then  $|G(P_{\min})|$  is used for  $G(P) < |G(P_{\min})|$ .

## Phonon Scattering

The phonon-scattering portion of [Equation 291](#) also has 2D and 3D parts. The way in which these parts are combined depends on the value of the `PhononCombination` parameter in the `IAlMob` command file:

$$\begin{aligned} \min \frac{\mu_{\text{ph},2D}}{D}, \mu_{\text{ph},3D} &\rightarrow , \text{ PhononCombination}=0 \\ \mu_{\text{ph}} = & \left[ \frac{D}{\mu_{\text{ph},2D}} + \frac{1}{\mu_{\text{ph},3D}} \right]^{-1} , \text{ PhononCombination}=1 \text{ (default)} \\ & \left[ \frac{D}{\mu_{\text{ph},2D}} + \frac{f(F_{\perp}, t, T)}{\mu_{\text{ph},3D}} \right]^{-1} , \text{ PhononCombination}=2 \end{aligned} \quad (307)$$

where:

$$\mu_{\text{ph},2D} = \frac{B}{F_{\perp}} + \frac{C \alpha_{\text{ph},2D,A}((N_{A,0} + N_2/2)/1 \text{ cm}^{-3})^{\lambda_{\text{ph},2D,A}} + \alpha_{\text{ph},2D,D}((N_{D,0} + N_2/2)/1 \text{ cm}^{-3})^{\lambda_{\text{ph},2D,D}}}{F_{\perp}^{1/3} (T/300 \text{ K})^k} \quad (308)$$

$$\mu_{\text{ph},3D} = \mu_{\text{max}} \frac{T}{300 \text{ K}}^{-\theta} \quad (309)$$

## Surface Roughness Scattering

Surface roughness scattering is given by:

$$\mu_{\text{sr}} = \frac{\alpha_{\text{sr},A}((N_{A,0} + N_2/2)/1 \text{ cm}^{-3})^{\lambda_{\text{sr},A}} + \alpha_{\text{sr},D}((N_{D,0} + N_2/2)/1 \text{ cm}^{-3})^{\lambda_{\text{sr},D}}}{\frac{(F_{\perp}/1 \text{ V/cm})^{A^*}}{\delta} + \frac{F_{\perp}^3}{\eta}} \quad (310)$$

where:

$$A^* = A + \frac{\alpha_{\perp}(n+p)}{((N_{A,0} + N_{D,0} + N_1)/1 \text{ cm}^{-3})^v} \quad (311)$$

## Parameters

Parameters associated with the model are accessible in the `IALMob` parameter set. Their values for silicon are shown in [Table 62](#) and [Table 63](#).

*Table 62 IALMob parameters (part 1): Default coefficients for silicon*

Symbol	Parameter name	Value	Unit
—	EnormMinimum	0.0	V/cm
$N_{D,\text{ref}}$	nref_D	$4 \times 10^{20}$	$\text{cm}^{-3}$
$N_{A,\text{ref}}$	nref_A	$7.2 \times 10^{20}$	$\text{cm}^{-3}$
$c_D$	cref_D	0.21	1
$c_A$	cref_A	0.5	1
$m_e^*/m_0$	me_over_m0	1.0	1
$m_h^*/m_0$	mh_over_m0	1.258	1

*Table 63 IALMob parameters (part 2): Default coefficients for silicon*

Symbol	Parameter name	Electron value	Hole value	Unit
$\mu_{\text{max}}$	umax	1417.0	470.5	$\text{cm}^2/(\text{Vs})$
$\mu_{\text{min}}$	umin	52.2	44.9	$\text{cm}^2/(\text{Vs})$
$\theta$	theta	2.285	2.247	1
$N_{\text{ref}}$	n_ref	$9.68 \times 10^{16}$	$2.23 \times 10^{17}$	$\text{cm}^{-3}$
$N_{\text{dop,ref}}$	ndop_ref	$10^{18}$	$10^{18}$	$\text{cm}^{-3}$
$N_{\text{sc,ref}}$	nsc_ref	$10^{18}$	$10^{18}$	$\text{cm}^{-3}$
$\alpha$	alpha	0.68	0.719	1

**Chapter 15: Mobility**  
Mobility Degradation at Interfaces

*Table 63 IALMob parameters (part 2): Default coefficients for silicon (Continued)*

<b>Symbol</b>	<b>Parameter name</b>	<b>Electron value</b>	<b>Hole value</b>	<b>Unit</b>
<i>S</i>	<i>s</i>	0.3042	0.3042	$\text{K}(\text{cm}/\text{V})^{2/3}$
<i>S<sub>t</sub></i>	<i>s_t</i>	0.0	0.0	$\text{K}(\mu\text{m})^2$
<i>t<sub>0</sub></i>	<i>t_0</i>	0.0005	0.0005	$\mu\text{m}$
<i>p</i>	<i>p</i>	4.0	4.0	1
<i>l<sub>crit,C</sub></i>	<i>l_crit_c</i>	$10^3$	$10^3$	cm
<i>B</i>	<i>B</i>	$9.0 \times 10^5$	$9.0 \times 10^5$	cm/s
<i>C</i>	<i>C</i>	4400.0	4400.0	$\text{cm}^{5/3}/\text{V}^{2/3}/\text{s}$
<i>λ</i>	<i>lambda</i>	0.057	0.057	1
<i>k</i>	<i>k</i>	1.0	1.0	1
$\alpha_{\text{ph},2\text{D,A}}$	<i>alpha_ph2d_A</i>	1.0	1.0	1
$\alpha_{\text{ph},2\text{D,D}}$	<i>alpha_ph2d_D</i>	1.0	1.0	1
$\lambda_{\text{ph},2\text{D,A}}$	<i>lambda_ph2d_A</i>	1.0	1.0	1
$\lambda_{\text{ph},2\text{D,D}}$	<i>lambda_ph2d_D</i>	1.0	1.0	1
<i>δ</i>	<i>delta</i>	$3.97 \times 10^{13}$	$3.97 \times 10^{13}$	$\text{cm}^2/(\text{Vs})$
$\lambda_{\text{sr}}$	<i>lambda_sr</i>	0.057	0.057	1
<i>A</i>	<i>A</i>	2.0	2.0	1
$\alpha_{\perp}$	<i>alpha_sr</i>	0.0	0.0	$\text{cm}^3$
<i>v</i>	<i>nu</i>	0.0	0.0	1
<i>η</i>	<i>eta</i>	$1.0 \times 10^{50}$	$1.0 \times 10^{50}$	$\text{V}^2/\text{cm/s}$

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

**Table 63 IALMob parameters (part 2): Default coefficients for silicon (Continued)**

Symbol	Parameter name	Electron value	Hole value	Unit
$N_1$	N1	1.0	1.0	$\text{cm}^{-3}$
$N_2$	N2	1.0	1.0	$\text{cm}^{-3}$
$\alpha_{\text{sr},A}$	alpha_sr_A	1.0	1.0	1
$\alpha_{\text{sr},D}$	alpha_sr_D	1.0	1.0	1
$\lambda_{\text{sr},A}$	lambda_sr_A	1.0	1.0	1
$\lambda_{\text{sr},D}$	lambda_sr_D	1.0	1.0	1
$l_{\text{crit}}$	l_crit	$10^3$	$10^3$	cm
$D_{1,\text{inv}}$	D1_inv	135.0	135.0	$\text{cm}^2/(\text{Vs})$
$D_{2,\text{inv}}$	D2_inv	40.0	40.0	$\text{cm}^2/(\text{Vs})$
$v_{0,\text{inv}}$	nu0_inv	1.5	1.5	1
$v_{1,\text{inv}}$	nul_inv	2.0	2.0	1
$v_{2,\text{inv}}$	nu2_inv	0.5	0.5	1
$\alpha_{1,\text{inv}}$	alphal_inv	0.0	0.0	1
$\alpha_{2,\text{inv}}$	alpha2_inv	0.0	0.0	1
$D_{1,\text{acc}}$	D1_acc	135.0	135.0	$\text{cm}^2/(\text{Vs})$
$D_{2,\text{acc}}$	D2_acc	40.0	40.0	$\text{cm}^2/(\text{Vs})$
$v_{0,\text{acc}}$	nu0_acc	1.5	1.5	1
$v_{1,\text{acc}}$	nul_acc	2.0	2.0	1
$v_{2,\text{acc}}$	nu2_acc	0.5	0.5	1

**Table 63 IALMob parameters (part 2): Default coefficients for silicon (Continued)**

Symbol	Parameter name	Electron value	Hole value	Unit
$\alpha_{1,acc}$	alphal_acc	0.0	0.0	1
$\alpha_{2,acc}$	alpha2_acc	0.0	0.0	1
$t_{Coulomb}$	tcoulomb	0.0	0.0	μm
$t_1$	t1	0.0003	0.0003	μm
$a_{ph,2D}$	a_ph2d	1.0	1.0	1
$a_{ph,3D}$	a_ph3d	1.0	1.0	1
$a_{C,2D}$	a_c2d	1.0	1.0	1
$a_{C,3D}$	a_c3d	1.0	1.0	1
$a_{sr}$	a_sr	1.0	1.0	1

## Stress Factors for Mobility Components

If isotropic stress-dependent mobility enhancement factors are applied to mobility components (see [Factor Models Applied to Mobility Components on page 995](#)), the IALMob mobility components become:

$$\begin{aligned}
 \mu'_{ph,2D} &= \gamma_{ph,2D} \mu_{ph,2D}, \quad \gamma_{ph,2D} = 1 + a_{ph,2D}(\gamma - 1) \\
 \mu'_{ph,3D} &= \gamma_{ph,3D} \mu_{ph,3D}, \quad \gamma_{ph,3D} = 1 + a_{ph,3D}(\gamma - 1) \\
 \mu'_{C,2D} &= \gamma_{C,2D} \mu_{C,2D}, \quad \gamma_{C,2D} = 1 + a_{C,2D}(\gamma - 1) \\
 \mu'_{C,3D} &= \gamma_{C,3D} \mu_{C,3D}, \quad \gamma_{C,3D} = 1 + a_{C,3D}(\gamma - 1) \\
 \mu'_{sr} &= \gamma_{sr} \mu_{sr}, \quad \gamma_{sr} = 1 + a_{sr}(\gamma - 1)
 \end{aligned} \tag{312}$$

where  $a_{ph,2D}$ ,  $a_{ph,3D}$ ,  $a_{C,2D}$ ,  $a_{C,3D}$ , and  $a_{sr}$  are stress scaling parameters that can be specified in the IALMob parameter set, and  $\gamma$  is an isotropic enhancement factor calculated from a stress-dependent Factor model (see [Isotropic Factor Models on page 988](#)).

In [Equation 308](#) and [Equation 309](#), exponents of the temperature for 2D phonons ( $k$ ) and 3D phonons ( $\theta$ ) are constant, which correspond to the default behavior of the model. For a more accurate treatment of mobility as a function of temperature (particularly, mobility at low

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

temperatures), the exponents become a function of temperature itself. For more details, see [Low-Temperature Phonon Exponent Correction for Mobility Models on page 435](#) and [Inversion and Accumulation Layer Mobility Model on page 438](#).

## Using the Inversion and Accumulation Layer Mobility Model

The inversion and accumulation layer mobility model is selected by specifying the `IALMob` keyword as an argument to `Enormal` in the command file. No mobility `DopingDependence` should be specified, as this is already included in the model. It is also not necessary to specify `-ConstantMobility`, as this will be invoked automatically with `IALMob`.

For small values of  $F_{\perp}$ , the acoustic phonon and surface roughness components of `IALMob` make an insignificant contribution to the total mobility. If required, the parameter `EnormMinimum` can be specified (in the parameter file) to suppress the calculation of acoustic phonon scattering and surface roughness for  $F_{\perp} < \text{EnormMinimum}$ .

The complete model described in [13] is obtained by combining `IALMob` with the Hänsch model [14] for high-field saturation (see [Extended Canali Model on page 440](#)). For example:

```
Physics {
    Mobility (
        Enormal( IALMob )
        HighFieldSaturation
    )
}
```

To select the Hänsch model, specify the parameter  $\alpha = 1$  in the `HighFieldDependence` section. In addition, the  $\beta$  exponent in the Hänsch model is equal to 2, which can be specified by setting  $\beta_0 = 2$  and  $\beta_{\text{exp}} = 0$ :

```
HighFieldDependence {
    alpha    = 1.0, 1.0          # [1]
    beta0   = 2.0, 2.0          # [1]
    betaexp = 0.0, 0.0
}
```

## Named Parameter Sets for the IALMob Model

The `IALMob` parameter set can be named. For example, in the parameter file, you can write the following to declare a parameter set with the name `myset`:

```
IALMob "myset" { ... }
```

To use a named parameter set, specify its name with `ParameterSetName` as an option to `IALMob` in the command file. For example:

```
Mobility (
    Enormal( IALMob( ParameterSetName = "myset" ... ) ... )
)
```

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

By default, the unnamed parameter set is used.

## Auto-Orientation for the IALMob Model

The IALMob model supports the auto-orientation framework (see [Auto-Orientation Framework on page 86](#)) that switches between different named parameter sets based on the orientation of the nearest interface. This can be activated by specifying the IALMob command file parameter AutoOrientation:

```
Mobility (
    Enormal( IALMob( AutoOrientation ... ) ... )
)
```

---

## University of Bologna Surface Mobility Model

The University of Bologna surface mobility model was developed for an extended temperature range between 25°C and 648°C. It should be used together with the University of Bologna bulk mobility model (see [University of Bologna Bulk Mobility Model on page 390](#)). The inversion layer mobility in MOSFETs is degraded by Coulomb scattering at low normal fields and by surface phonons and surface roughness scattering at large normal fields.

In the University of Bologna model [4], all these effects are combined by using Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{bsc}} + \frac{D}{\mu_{ac}} + \frac{D}{\mu_{sr}} \quad (313)$$

where  $1/\mu_{bsc}$  is the contribution of Coulomb scattering, and  $1/\mu_{ac}$  and  $1/\mu_{sr}$  are those of surface phonons and surface roughness scattering, respectively.  $D = \exp(-x/l_{crit})$  (where  $x$  is the distance from the interface and  $l_{crit}$  a fit parameter) is a damping that switches off the inversion layer terms far away from the interface.

The term  $\mu_{bsc}$  is associated with substrate impurity and carrier concentration. It is decomposed in an unscreened part (due to the impurities) and a screened part (due to local excess carrier concentration):

$$\mu_{bsc}^{-1} = \mu_b^{-1} \left[ D \frac{1}{(1 + f_{sc}^{\tau})^{1/\tau}} - 1 + D_b \right] \quad (314)$$

where  $\mu_b$  is given by the bulk mobility model, and  $\tau$  is a fit parameter. The screening function is given by:

$$f_{sc} = \frac{N_1}{N_{A,0} + N_{D,0}} \frac{\eta}{N_{A,0} + N_{D,0}} \quad (315)$$

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

where  $N_{np}$  is a total carrier concentration which exceeds the electron and hole equilibrium concentrations ( $n_{eq}$  and  $p_{eq}$ ) and is computed as follows:

$$N_{np} = C_n^\alpha(n - n_{eq}) + C_p^\alpha(p - p_{eq}) \quad (316)$$

where  $C_{n,p}^\alpha$  are electron and hole concentration coefficients used in the mobility for a carrier  $\alpha$  (electron or hole).

If surface mobility is plotted against the effective normal field, mobility data converges toward a universal curve. Deviations from this curve appear at the onset of weak inversion, and the threshold field changes with the impurity concentration at the semiconductor surface [15]. The term  $\mu_{bsc}$  models these deviations, in that, it is the roll-off in the effective mobility characteristics. As the effective field increases, the mobilities become independent of the channel doping and approach the universal curve.

The main scattering mechanisms are, in this case, surface phonons and surface roughness scattering, which are expressed by:

$$\mu_{ac} = C(T) \frac{N_{A,0} + N_{D,0}}{N_2}^a \frac{1}{F_\perp^\delta} \quad (317)$$

$$\mu_{sr} = B(T) \frac{N_{A,0} + N_{D,0} + N_3}{N_4}^b \frac{1}{F_\perp^\lambda} \quad (318)$$

$F_\perp$  is the electric field normal to the semiconductor–insulator interface (see [Computing Transverse Field on page 426](#)).

Parameters for the model are accessible in the `UniBoEnormalDependence` parameter set. [Table 64](#) lists the silicon default parameters. The reported parameters are detailed in the literature [16]. The model was calibrated with experiments [15][16] in the temperature range from 300 K to 700 K.

*Table 64 Parameters of University of Bologna surface mobility model ( $T_n = T/300$  K)*

Symbol	Parameter	Electrons	Holes	Unit
$N_1$	<code>N1</code>	$2.34 \times 10^{16}$	$2.02 \times 10^{16}$	$\text{cm}^{-3}$
$N_2$	<code>N2</code>	$4.0 \times 10^{15}$	$7.8 \times 10^{15}$	$\text{cm}^{-3}$
$N_3$	<code>N3</code>	$1.0 \times 10^{17}$	$2.0 \times 10^{15}$	$\text{cm}^{-3}$
$N_4$	<code>N4</code>	$2.4 \times 10^{18}$	$6.6 \times 10^{17}$	$\text{cm}^{-3}$
$B$	<code>B</code>	$5.8 \times 10^{18} T_n^{\gamma_B}$	$7.82 \times 10^{15} T_n^{\gamma_B}$	$\text{cm}^2/\text{Vs}$

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

Table 64 Parameters of University of Bologna surface mobility model ( $T_n = T/300\text{ K}$ )

Symbol	Parameter	Electrons	Holes	Unit
$\gamma_B$	B_exp	0	1.4	1
$C$	C	$1.86 \times 10^4 T_n^{-\gamma_C}$	$5.726 \times 10^3 T_n^{-\gamma_C}$	$\text{cm}^2/\text{Vs}$
$\gamma_C$	C_exp	2.1	1.3	1
$\tau$	tau	1	3	1
$C_{n,p}^{electron}$	c_ele	1	0	1
$C_{n,p}^{hole}$	c_hole	0	1	1
$\eta$	eta	0.3	0.5	1
$a$	ac_exp	0.026	-0.02	1
$b$	sr_exp	0.11	0.08	1
$l_{crit}$	l_crit	$1 \times 10^{-6}$	$1 \times 10^{-6}$	cm
$\delta$	delta	0.29	0.3	1
$\lambda$	lambda	2.64	2.24	1
$D_b$	a	0	0	1

#### Note:

Equation 317 and Equation 318 have a root dependence on  $F_{\perp}$  and, for a relatively small  $F_{\perp}$ , this might lead to numeric instabilities. To resolve that, you can specify `MobEnormalUniboRegularizationField=1000 [V/cm]` in the `Math` section. Simulation results are not affected by this setting if practically useful  $F_{\perp}$  is higher than the regularization field defined by such a setting.

## Mobility Degradation Components due to Coulomb Scattering

Different mobility degradation components due to Coulomb scattering are available in Sentaurus Device that can be combined with other interface mobility degradation models:

- `NegInterfaceCharge` accounts for mobility degradation due to a negative interface charge (from charged traps and fixed charge).
- `PosInterfaceCharge` accounts for mobility degradation due to a positive interface charge (from charged traps and fixed charge).
- `Coulomb2D` accounts for mobility degradation due to ionized impurities near the interface.

These degradation components can be specified separately or in combination with each other. If specified, they will be combined using Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{other}}} + \frac{1}{\mu_{\text{nic}}} + \frac{1}{\mu_{\text{pic}}} + \frac{1}{\mu_{\text{C2D}}} \quad (319)$$

where:

- $\mu_{\text{other}}$  represents the other `DopingDependence` and `Enormal` models specified for the simulation.
- $\mu_{\text{nic}}$  represents the `NegInterfaceCharge` mobility degradation component.
- $\mu_{\text{pic}}$  represents the `PosInterfaceCharge` mobility degradation component.
- $\mu_{\text{C2D}}$  represents the `Coulomb2D` mobility degradation component.

The general form of the mobility degradation components due to Coulomb scattering is given by:

$$\mu_C = \frac{\mu_1 \left( \frac{T}{300 \text{ K}} \right)^k \cdot 1 + \left[ c / c_{\text{trans}} \cdot \frac{N_{A,D} + N_1}{10^{18} \text{ cm}^{-3}} \cdot \frac{N_{\text{coulomb}}^{\gamma_1}}{N_0} \right]^{\eta_1} }{\frac{N_{A,D} + N_2}{10^{18} \text{ cm}^{-3}} \cdot \frac{N_{\text{coulomb}}^{\gamma_2}}{N_0} \cdot D \cdot f(F_{\perp})} \quad (320)$$

where:

- $N_{\text{coulomb}}$  = - negative interface charge density, for `NegInterfaceCharge` component  
positive interface charge density, for `PosInterfaceCharge` component  
local  $N_{A,D}$ , for `Coulomb2D` component
- $N_0$  = - Interface charge density, for `Neg/PosInterfaceCharge` components  
Bulk charge density, for `Coulomb2D` component

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

- $c = n$  (electron mobility) or  $p$  (hole mobility)
- $N_{A,D} = N_{A,0}$  (electron mobility) or  $N_{D,0}$  (hole mobility)

and:

$$f(F_\perp) = 1 - \exp[-(F_\perp/E_0)^\gamma] \quad (321)$$

$$D = \exp(-x/l_{\text{crit}}) \quad (322)$$

In Equation 322,  $x$  is the distance from the interface.

Parameters associated with the components are accessible in the parameter sets

`NegInterfaceChargeMobility`, `NegInterfaceChargeMobility_aniso`,

`PosInterfaceChargeMobility`, `PosInterfaceChargeMobility_aniso`,

`Coulomb2DMobility`, and `Coulomb2DMobility_aniso`. [Table 65](#) lists their default values.

**Table 65** Parameters for mobility degradation components due to Coulomb scattering

Symbol	Parameter name	$\mu_{\text{nic}}$ <b>Electron</b>	$\mu_{\text{nic}}$ <b>Hole</b>	$\mu_{\text{pic}}$ <b>Electron</b>	$\mu_{\text{pic}}$ <b>Hole</b>	$\mu_{\text{C2D}}$ <b>Electron</b>	$\mu_{\text{C2D}}$ <b>Hole</b>	Unit
$\mu_1$	<code>mul</code>	40.0	40.0	40.0	40.0	40.0	40.0	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
$k$	<code>T_exp</code>	1.0	1.0	1.0	1.0	1.0	1.0	1
$c_{\text{trans}}$	<code>c_trans</code>	$10^{18}$	$10^{18}$	$10^{18}$	$10^{18}$	$10^{18}$	$10^{18}$	$\text{cm}^{-3}$
$v$	<code>c_exp</code>	1.5	1.5	1.5	1.5	1.5	1.5	1
$\eta_1$	<code>Nc_exp1</code>	1.0	1.0	1.0	1.0	1.0	1.0	1
$\eta_2$	<code>Nc_exp2</code>	0.5	0.5	0.5	0.5	0.5	0.5	1
$N_0$	<code>N0</code>	$10^{11}$	$10^{11}$	$10^{11}$	$10^{11}$	$10^{11}$	$10^{11}$	$\text{cm}^{-2}$
		$10^{18}$	$10^{18}$	$10^{18}$	$10^{18}$	$10^{18}$	$10^{18}$	$\text{cm}^{-3}$
$N_1$	<code>N1</code>	1.0	1.0	1.0	1.0	1.0	1.0	$\text{cm}^{-3}$
$N_2$	<code>N2</code>	1.0	1.0	1.0	1.0	1.0	1.0	$\text{cm}^{-3}$
$\gamma_1$	<code>N_exp1</code>	0.0	0.0	0.0	0.0	0.0	0.0	1
$\gamma_2$	<code>N_exp2</code>	0.0	0.0	0.0	0.0	0.0	0.0	1

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

Table 65 Parameters for mobility degradation components due to Coulomb scattering

Symbol	Parameter name	$\mu_{\text{nic}}$ Electron	$\mu_{\text{nic}}$ Hole	$\mu_{\text{pic}}$ Electron	$\mu_{\text{pic}}$ Hole	$\mu_{\text{C2D}}$ Electron	$\mu_{\text{C2D}}$ Hole	Unit
$l_{\text{crit}}$	l_crit	$10^{-6}$	$10^{-6}$	$10^{-6}$	$10^{-6}$	$10^{-6}$	$10^{-6}$	cm
$E_0$	E0	$2.0 \times 10^5$	$2.0 \times 10^5$	$2.0 \times 10^5$	$2.0 \times 10^5$	$2.0 \times 10^5$	$2.0 \times 10^5$	V/cm
$\gamma$	En_exp	2.0	2.0	2.0	2.0	2.0	2.0	1
$a_C$	a_c	1.0	1.0	1.0	1.0	1.0	1.0	1

## Stress Factors for Mobility Components

If isotropic stress-dependent mobility enhancement factors are applied to mobility components (see [Factor Models Applied to Mobility Components on page 995](#)), the Coulomb degradation component becomes:

$$\mu'_C = \gamma_C \mu_C, \quad \gamma_C = 1 + a_C(\gamma - 1) \quad (323)$$

where  $a_C$  is a stress scaling parameter that can be specified in the NegInterfaceChargeMobility, PosInterfaceChargeMobility, or Coulomb2DMobility parameter set, and  $\gamma$  is an isotropic enhancement factor calculated from a stress-dependent Factor model (see [Isotropic Factor Models on page 988](#)).

## Using Mobility Degradation Components

The Coulomb2D model is a local model that uses local values of  $N_A$  and  $N_D$ . To use this model, specify Coulomb2D in addition to the standard Enormal model. For example:

```
Physics {
    Mobility (
        PhuMob HighFieldSaturation
        Enormal (Lombardi Coulomb2D)
    )
}
```

The NegInterfaceCharge and PosInterfaceCharge models are nonlocal models because  $N_C$  used in these models is a charge density at a location that might be different from the point where mobility is being calculated.

If Math{-GeometricDistances} is not specified,  $N_{\text{coulomb}}$  is the charge density at the point on the interface that is the closest distance to the point where mobility is being calculated. If

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

there is not a vertex at this interface point,  $N_{\text{coulomb}}$  is interpolated from the charge density at the surrounding vertices.

If `Math{-GeometricDistances}` is specified,  $N_{\text{coulomb}}$  is the charge density at the vertex on the interface that is the closest distance to the point where mobility is being calculated.

To use these models, specify `NegInterfaceCharge`, or `PosInterfaceCharge`, or both in addition to a standard `Enormal` model. For convenience, both models can be specified with the single keyword `InterfaceCharge`. For example:

```
Physics {
    Mobility (
        PhuMob HighFieldSaturation
        Enormal (Lombardi InterfaceCharge)
    )
}
```

#### Note:

When using mobility degradation components, a standard `Enormal` model (such as `Lombardi`) must be specified explicitly in addition to the mobility degradation component. Sentaurus Device will not include the `Lombardi` model by default (this only occurs when `Enormal` is specified with no arguments).

By default, the `NegInterfaceCharge` and `PosInterfaceCharge` models use the charge density  $N_{\text{coulomb}}$  located at the nearest semiconductor-insulator interface. Alternatively, these models can use the charge density at the nearest user-defined surface by specifying the surface name as an argument to the interface charge model. For example:

```
Physics {
    Mobility (
        PhuMob HighFieldSaturation
        Enormal (Lombardi
            NegInterfaceCharge(SurfaceName="S1")
            PosInterfaceCharge(SurfaceName="S2")
            ...
        )
    )
}
```

Surfaces are defined in the global `Math` section and represent the union of an arbitrary number of interfaces. The following example specifies a surface named `s1` that consists of all the `HfO2-oxide` interfaces, as well as the `region_1-region_2` interface:

```
Math {
    Surface "S1" (
        MaterialInterface="HfO2/Oxide"
        RegionInterface="region_1/region_2"
    )
}
```

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

## Remote Coulomb Scattering Model

High-k gate dielectrics are being considered as an alternative to SiO<sub>2</sub> to reduce unacceptable leakage currents as transistor dimensions decrease. One obstacle when using high-k gate dielectrics is that a degraded carrier mobility is often observed for such devices. Although the causes of high-k mobility degradation are not completely understood, a possible contributor is remote Coulomb scattering (RCS).

Sentaurus Device provides an empirical model for RCS degradation that can be combined with other `Enormal` models using Matthiessen's rule:

$$\frac{1}{\mu_{\text{Enormal}}} = \frac{1}{\mu_{\text{Enormal}_1}} + \frac{1}{\mu_{\text{Enormal}_2}} + \dots + \frac{D_{\text{rcs}} D_{\text{rcs\_highk}}}{\mu_{\text{rcs}}} \quad (324)$$

The `RCS` model is taken from [17] and accounts for the mobility degradation observed with HfSiON MISFETs. This is attributed to RCS:

$$\mu_{\text{rcs}} = \mu_{\text{rcs}0} \frac{\frac{N_{A,D}}{3 \times 10^{16} \text{ cm}^{-3}}^{\gamma_1} \frac{T}{300 \text{ K}}^{\gamma_2} (g_{\text{screening}})^{\gamma_3 + \gamma_4 \cdot \ln \frac{N_{A,D}}{3 \times 10^{16} \text{ cm}^{-3}}}^{\gamma_3 + \gamma_4 \cdot \ln \frac{N_{A,D}}{3 \times 10^{16} \text{ cm}^{-3}}} / f(F_{\perp}) \quad (325)$$

where  $N_{A,D} = N_{A,0}$  (electron mobility) or  $N_{D,0}$  (hole mobility).

In Equation 325, the  $g_{\text{screening}}$  factor accounts for screening of the remote charge by carriers. In [17], this is expressed in terms of the inversion charge density. Here, a local expression that depends on carrier concentration is used:

$$g_{\text{screening}} = s + \frac{c}{c_0 \frac{N_{A,D}}{3 \times 10^{16} \text{ cm}^{-3}}^{\gamma_5}} \quad (326)$$

In Equation 326,  $c$  is the carrier concentration ( $n$  for electron mobility and  $p$  for hole mobility), and  $s$ ,  $c_0$ , and  $\gamma_5$  are parameters.

In Equation 325,  $f(F_{\perp})$  is a function that confines the degradation to areas of the structure where  $F_{\perp}$  is large enough to initiate inversion:

$$f(F_{\perp}) = 1 - \exp(-\xi F_{\perp} / N_{\text{depl}}) \quad (327)$$

In the above expression,  $N_{\text{depl}}$  is a doping- and temperature-dependent approximation for the depletion charge density [ $\text{cm}^{-2}$ ].

The distance factors used in Equation 324 are given by:

$$D_{\text{rcs}} = \exp(-(\text{dist} + d_{\text{crit}}) / l_{\text{crit}}) \quad (328)$$

$$D_{\text{rcs\_highk}} = \exp(-\text{dist}_{\text{highk}} / l_{\text{crit\_highk}}) \quad (329)$$

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

In these expressions,  $\text{dist}$  is the distance from the nearest semiconductor–insulator interface, and  $\text{dist}_{\text{highk}}$  is the distance from the nearest high-k insulator. If no high-k insulator is found in the structure,  $D_{\text{rcs\_highk}} = 1$ .

Parameters used in the RCS model are accessible in the `RCSMobility` parameter set in the parameter file. Values for silicon are shown in [Table 66](#).

*Table 66 RCSMobility parameters: Default coefficients for silicon*

Symbol	Parameter name	Electron value	Hole value	Unit
$\mu_{\text{rcs}0}$	<code>murcs0</code>	149.0	149.0	$\text{cm}^2/\text{Vs}$
$\gamma_1$	<code>gamma1</code>	-0.23187	-0.23187	1
$\gamma_2$	<code>gamma2</code>	2.1	2.1	1
$\gamma_3$	<code>gamma3</code>	0.40	0.40	1
$\gamma_4$	<code>gamma4</code>	0.05	0.05	1
$\gamma_5$	<code>gamma5</code>	1.0	1.0	1
$s$	<code>s</code>	0.1	0.1	1
$c_0$	<code>c0</code>	$3.0 \times 10^{16}$	$3.0 \times 10^{16}$	$\text{cm}^{-3}$
$d_{\text{crit}}$	<code>d_crit</code>	0.0	0.0	cm
$l_{\text{crit}}$	<code>l_crit</code>	$1 \times 10^{-6}$	$1 \times 10^{-6}$	cm
$l_{\text{crit\_highk}}$	<code>l_crit_highk</code>	$1 \times 10^6$	$1 \times 10^6$	cm
$\xi$	<code>xi</code>	$1.3042 \times 10^7$	$1.3042 \times 10^7$	$\text{V}^{-1}\text{cm}^{-1}$
$a_{\text{rcs}}$	<code>a_rcs</code>	1.0	1.0	1

## Stress Factors for Mobility Components

If isotropic stress-dependent mobility enhancement factors are applied to mobility components (see [Factor Models Applied to Mobility Components on page 995](#)), the RCS degradation becomes:

$$\mu'_{\text{rcs}} = \gamma_{\text{rcs}} \mu_{\text{rcs}} , \quad \gamma_{\text{rcs}} = 1 + a_{\text{rcs}}(\gamma - 1) \quad (330)$$

where  $a_{\text{rcs}}$  is a stress scaling parameter that can be specified in the `RCSMobility` parameter set, and  $\gamma$  is an isotropic enhancement factor calculated from a stress-dependent Factor model (see [Isotropic Factor Models on page 988](#)).

## Chapter 15: Mobility

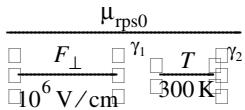
### Mobility Degradation at Interfaces

## Remote Phonon Scattering Model

Sentaurus Device also provides a simple empirical model for remote phonon scattering (RPS) degradation that can be combined with other `Enormal` models using Matthiessen's rule:

$$\frac{1}{\mu_{\text{Enormal}}} = \frac{1}{\mu_{\text{Enormal}_1}} + \frac{1}{\mu_{\text{Enormal}_2}} + \dots + \frac{D_{\text{rps}} D_{\text{rps\_highk}}}{\mu_{\text{rps}}} \quad (331)$$

The RPS model is extracted from figures in [18] and accounts for a portion of the mobility degradation observed with HfO<sub>2</sub>-gated MOSFETs. This term is attributed to RPS:

$$\mu_{\text{rps}} = \frac{\mu_{\text{rps}0}}{F_{\perp}^{\gamma_1} T^{\gamma_2}} \quad (332)$$


The distance factors used in [Equation 331](#) are given by:

$$D_{\text{rps}} = \exp(-(dist + d_{\text{crit}})/l_{\text{crit}}) \quad (333)$$

$$D_{\text{rps\_highk}} = \exp(-dist_{\text{highk}}/l_{\text{crit\_highk}}) \quad (334)$$

In these expressions, `dist` is the distance from the nearest semiconductor–insulator interface, and `disthighk` is the distance from the nearest high-k insulator. If no high-k insulator is found in the structure,  $D_{\text{rps\_highk}} = 1$ .

Parameters used in the RPS model are accessible in the `RPSMobility` parameter set in the parameter file. Values for silicon are shown in [Table 67](#).

*Table 67 RPSMobility parameters: Default coefficients for silicon*

Symbol	Parameter name	Electron value	Hole value	Unit
$\mu_{\text{rps}0}$	<code>murps0</code>	496.7	496.7	$\text{cm}^2/\text{Vs}$
$\gamma_1$	<code>gamma1</code>	0.68	0.68	1
$\gamma_2$	<code>gamm2</code>	0.34	0.34	1
$d_{\text{crit}}$	<code>d_crit</code>	0.0	0.0	cm
$l_{\text{crit}}$	<code>l_crit</code>	$1 \times 10^{-6}$	$1 \times 10^{-6}$	cm
$l_{\text{crit\_highk}}$	<code>l_crit_highk</code>	$1 \times 10^6$	$1 \times 10^6$	cm
$a_{\text{rps}}$	<code>a_rps</code>	1.0	1.0	1

## Chapter 15: Mobility

### Mobility Degradation at Interfaces

## Stress Factors for Mobility Components

If isotropic stress-dependent mobility enhancement factors are applied to mobility components (see [Factor Models Applied to Mobility Components on page 995](#)), the RPS degradation becomes:

$$\mu'_{\text{rps}} = \gamma_{\text{rps}} \mu_{\text{rps}} , \quad \gamma_{\text{rps}} = 1 + a_{\text{rps}}(\gamma - 1) \quad (335)$$

where  $a_{\text{rps}}$  is a stress scaling parameter that can be specified in the `RPSMobility` parameter set, and  $\gamma$  is an isotropic enhancement factor calculated from a stress-dependent Factor model (see [Isotropic Factor Models on page 988](#)).

---

## Computing Transverse Field

Sentaurus Device supports two different methods for computing the normal electric field  $F_{\perp}$ :

- Using the normal to the interface
- Using the normal to the current

For vertices at the interface, an optional correction  $F_{\text{corr}}$  for the field value is also available.

## Normal to the Interface

Assume that mobility degradation occurs at an interface  $\Gamma$ . By default, the distance to the interface is the true geometric distance, and  $\hat{n}$  is the gradient of the distance to the interface. When the `-GeometricDistances` option is specified in the `Math` section, for a given point  $\vec{r}$ , Sentaurus Device locates the nearest vertex  $\vec{r}_i$  on the interface  $\Gamma$ , approximates the distance of  $\vec{r}$  to the interface by the distance to  $\vec{r}_i$ , and determines the direction  $\hat{n}$  normal to the interface at vertex  $\vec{r}_i$ . From  $\hat{n}$ , the normal electric field is:

$$F_{\perp}(\vec{r}) = \left| \vec{F}(\vec{r}) \cdot \hat{n} + F_{\text{corr}} \right| \quad (336)$$

To activate the Lombardi model with this method of computing  $F_{\perp}$ , specify the `Enormal` flag to `Mobility`. The keyword `ToInterfaceEnormal` is synonymous with `Enormal`.

By default, the interface  $\Gamma$  is a semiconductor–insulator interface. Sometimes, it is important to change this default interface definition, for example, when the insulator (oxide) is considered a wide-bandgap semiconductor. In this case, Sentaurus Device allows this interface to be specified with `EnormalInterface` in the `Math` section.

In the following example, Sentaurus Device takes as  $\Gamma$  the union of interfaces between materials, `OxideAsSemiconductor` and `Silicon`, and regions, `regionK1` and `regionL1`:

```
Math {
    EnormalInterface (
        regioninterface= [ "regionK1/regionL1" ] ,
        materialinterface= [ "OxideAsSemiconductor/Silicon" ]
```

## Chapter 15: Mobility

### Thin-Layer Mobility Model

```
}
```

## Normal to the Current Flow

Using this method,  $F_{\perp}(\vec{r})$  is defined as the component of the electric field normal to the electron ( $c = n$ ) or hole ( $c = p$ ) currents  $\vec{J}_c(\vec{r})$ :

$$F_{c,\perp}(\vec{r}) = F(\vec{r}) \sqrt{1 - \frac{\vec{F}(\vec{r}) \cdot \vec{J}_c(\vec{r})}{|F(\vec{r}) J_c(\vec{r})|^2}} + F_{\text{corr}} \quad (337)$$

where  $F_{n,\perp}$  is used for the evaluation of electron mobility and  $F_{p,\perp}$  is used for the evaluation of hole mobility. Through corrections of the current,  $F_{n,\perp}$  and  $F_{p,\perp}$  also are affected by the keyword `ParallelToInterfaceInBoundaryLayer` (see [Field Correction Close to Interfaces on page 452](#)).

#### Note:

For very low current levels, the computation of the electric field component normal to the currents might be numerically problematic and lead to convergence problems. It is recommended to use the option `Enormal`. Besides possible numeric problems, both approaches give the same or very similar results.

## Field Correction on Interface

For vertices on the interface, due to discretization, the normal electric field in inversion is underestimated systematically due to screening by the charge at the same vertex.

Therefore, Sentaurus Device supports a correction of the field:

$$F_{\text{corr}}(\vec{r}) = \frac{\alpha l \rho(\vec{r})}{\epsilon} \quad (338)$$

where  $\alpha$  is dimensionless and is specified with `NormalFieldCorrection` in the `Math` section (reasonable values range from 0 to 1; the default is zero),  $\rho$  is the space charge,  $\epsilon$  is the dielectric constant in the semiconductor, and  $l$  is an estimate for the depth of the box of the vertex in the normal direction.

---

## Thin-Layer Mobility Model

The thin-layer mobility model applies to devices with silicon layers that are only a few nanometers thick. In such devices, geometric quantization leads to a mobility that cannot be expressed with a normal field-dependent interface model such as those described in [Mobility Degradation at Interfaces on page 403](#), but it depends explicitly on the layer thickness.

## Chapter 15: Mobility

### Thin-Layer Mobility Model

The thin-layer mobility model described here is based on the model described in the literature [19] and is used in conjunction with either the Lombardi model (see Enhanced Lombardi Model on page 404) or the IALMob model (see Inversion and Accumulation Layer Mobility Model on page 407).

#### Note:

When using the thin-layer mobility model in conjunction with the Lombardi model, it is recommended to also use the Philips Unified Mobility Model on page 398.

The Philips unified mobility model must be activated separately from the thin-layer mobility model.

The thin-layer mobility  $\mu_{tl}$  consists of contributions of thickness fluctuation scattering, surface phonon scattering, bulk phonon scattering, and additional contributions from the normal field-dependent interface model used in conjunction with it:

$$\frac{1}{\mu_{tl}} = \frac{D}{\mu_{tf}} + \frac{D}{\mu_{sp}} + \frac{D}{\mu_{bp}} + \frac{\frac{D}{\mu_{sr}}}{\frac{D}{\mu_{sr}} + \frac{1}{\mu_{ph,3D}} + \frac{1}{\mu_C}}, \begin{array}{l} \text{Lombardi} \\ \text{IALMob} \end{array} \quad (339)$$

where:

- $D = \exp(-x/l_{crit})$  (see Enhanced Lombardi Model on page 404 (Lombardi) or Inversion and Accumulation Layer Mobility Model on page 407 (IALMob)).
- $\mu_{sr}$  is given by Equation 286 (Lombardi) or Equation 310 (IALMob).
- $\mu_{ph,3D}$  is given by Equation 309 (IALMob).
- $\mu_C$  is given by Equation 295 (IALMob).

The thickness fluctuation term is given by:

$$\frac{1}{\mu_{tf}} = \frac{1}{\mu_{tf0}(t_b/1\text{ nm})^{\eta_1} \left[ 1 + \frac{F_{tf0}}{F_{tfh0}} \right]^{\eta_2}} + \frac{1}{\mu_{tfh0}(t_b/1\text{ nm})^{\eta_1} \frac{F_{tfh0}}{F_{\perp}}} \quad (340)$$

The surface phonon term is given by:

$$\mu_{sp} = \mu_{sp0} \exp(t_b/t_{sp0}) \quad (341)$$

The bulk phonon term is given by:

$$\mu_{bp} = P_1 \mu_{ac,1} + (1 - P_1) \mu_{ac,2} \quad (342)$$

$$P_1 = p_1 + \frac{1 - p_1}{1 + p_2 \exp(-p_3 \Delta E/kT)} \quad (343)$$

## Chapter 15: Mobility

### Thin-Layer Mobility Model

$$\Delta E = \frac{\hbar^2 \pi^2}{2t_b^2} \frac{1}{m_{z2}} - \frac{1}{m_{z1}} \quad (344)$$

$$\mu_{ac,v} = \frac{\mu_{ac0,v}}{[1 + (W_{Tv}/W_{Fv})^\beta]^{1/\beta}} \quad (345)$$

$$W_{Tv} = \frac{2}{3} t_b + W_{T0v} \frac{t_b}{1 \text{ nm}}^4 \frac{F_\perp}{1 \text{ MV/cm}} \quad (346)$$

$$\frac{W_{Fv}}{1 \text{ nm}} = \zeta^{v-1} \frac{\mu_{ac}}{P_{bulk} \mu_{ac0,1} + \zeta (1 - P_{bulk}) \mu_{ac0,2}} \quad (347)$$

where:

- $\mu_{ac}$  is given by [Equation 285 \(Lombardi\)](#) or by  $\mu_{ph,2D}$  [Equation 308 \(IALMob\)](#).
- $P_{bulk} = p_1 + (1 - p_1)/(1 + p_2)$ .
- The thickness  $t_b$  is the larger of the local layer thickness and  $t_{min}$ .
- $t_{min}$ ,  $\mu_{tf0}$ ,  $F_{tf0}$ ,  $\mu_{tfh0}$ ,  $F_{tfh0}$ ,  $\eta_1$ ,  $\eta_2$ ,  $\mu_{sp0}$ ,  $t_{sp0}$ ,  $\mu_{ac0,v}$ ,  $p_1$ ,  $p_2$ ,  $p_3$ ,  $m_{z1}$ ,  $m_{z2}$ ,  $\beta$ ,  $W_{T0v}$ , and  $\zeta$  are model parameters.

## Using the Thin-Layer Mobility Model

To activate the thin-layer mobility model, specify `ThinLayer(<parameters>)` as an option to `eMobility`, `hMobility`, or `Mobility`. The optional parameters are `<geo_parameters>` and `<physical_parameters>`.

The `ThinLayer` subcommand has the same `<geo_parameters>` as the `LayerThickness` command (see [Extracting Layer Thickness on page 379](#)).

## Physical Parameters

Specify `Lombardi` or `IALMob` (including any optional arguments for these models) as an option to `ThinLayer` to indicate which normal field-dependent interface model is used in conjunction with the thin-layer mobility calculations (`Lombardi` is the default). For example:

```
Physics {
    Mobility (
        PhuMob
        ThinLayer (Lombardi(...))
    )
}
```

or:

```
Physics {
    Mobility (
        ThinLayer (IALMob(...))
```

## Chapter 15: Mobility

### Thin-Layer Mobility Model

```
    )  
}
```

#### Note:

When using `ThinLayer(Lombardi)` or `ThinLayer(IALMob)`, do not specify `Enormal(Lombardi)` or `Enormal(IALMob)` in addition. This would result in double-counting the scattering mechanisms that are already accounted for (see [Equation 339](#)). However, `Enormal` must still be used to include other required degradation terms. For example:

```
Physics {  
    Mobility ( ThinLayer (IALMob(...))  
                Enormal (InterfaceCharge)  
            )  
}
```

When `ThinLayer(Lombardi(...))` is specified, the parameters to compute  $D$ ,  $\mu_{sr}$ , and  $\mu_{ac}$  are the same as used for the Lombardi model (see [Table 60 on page 405](#) and [Table 61 on page 406](#)). Synopsys considers the parameters in [Table 61](#) to be more suitable for the thin-layer mobility model; however, the defaults are still given in [Table 60](#).

When `ThinLayer(IALMob(...))` is specified, the parameters to compute  $D$ ,  $\mu_{sr}$ ,  $\mu_{ph,3D}$ ,  $\mu_C$ , and  $\mu_{ph,2D}$  are the same as used for the `IALMob` model (see [Table 62 on page 411](#) and [Table 63 on page 411](#)).

The other parameters are specified as electron–hole pairs in the `ThinLayerMobility` parameter set. [Table 68](#) summarizes the parameters.

*Table 68 Parameters for thin-layer mobility model*

Symbol	Parameter name	Electron	Hole	Unit
$t_{min}$	<code>tmin</code>	0.002	0.002	$\mu\text{m}$
$\mu_{tf0}$	<code>mutf0</code>	0.15	0.28	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
$F_{tf0}$	<code>ftf0</code>	6250	$10^{100}$	$\text{Vcm}^{-1}$
$\mu_{tfh0}$	<code>mutfh0</code>	$10^6$	$10^6$	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
$F_{tfh0}$	<code>ftfh0</code>	$10^{100}$	$10^{100}$	$\text{Vcm}^{-1}$
$\eta_1$	<code>eta1</code>	6	6	1
$\eta_2$	<code>eta2</code>	1	1	1

**Chapter 15: Mobility**  
Thin-Layer Mobility Model

*Table 68 Parameters for thin-layer mobility model (Continued)*

<b>Symbol</b>	<b>Parameter name</b>	<b>Electron</b>	<b>Hole</b>	<b>Unit</b>
$\mu_{sp0}$	musp0	$1.145 \times 10^{-8}$	$1.6 \times 10^{-10}$	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
$t_{sp0}$	tsp0	$10^{-4}$	$10^{-4}$	$\mu\text{m}$
$\mu_{ac0,1}$	muac01	315	30.2	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
$\mu_{ac0,2}$	muac02	6.4	69	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
$p_1$	p1	0.55	0	1
$p_2$	p2	400	0.66	1
$p_3$	p3	1.44	1	1
$m_{z1}$	mz1	0.916	0.29	$m_0$
$m_{z2}$	mz2	0.19	0.25	$m_0$
$\beta$	beta	4	4	1
$W_{T01}$	wt01	$3 \times 10^{-6}$	0	$\mu\text{m}$
$W_{T02}$	wt02	$3.5 \times 10^{-7}$	0	$\mu\text{m}$
$\zeta$	zeta	2.88	1.05	1
$a_{tf}$	a_tf	1.0	1.0	1
$a_{sp}$	a_sp	1.0	1.0	1
$a_{bp}$	a_bp	1.0	1.0	1

## Chapter 15: Mobility

### Thin-Layer Mobility Model

## Stress Factors for Mobility Components

If isotropic stress-dependent mobility enhancement factors are applied to mobility components (see [Factor Models Applied to Mobility Components on page 995](#)), the `ThinLayer` mobility components become:

$$\begin{aligned}\mu'_{tf} &= \gamma_{tf}\mu_{tf}, \quad \gamma_{tf} = 1 + a_{tf}(\gamma - 1) \\ \mu'_{sp} &= \gamma_{sp}\mu_{sp}, \quad \gamma_{sp} = 1 + a_{sp}(\gamma - 1) \\ \mu'_{bp} &= \gamma_{bp}\mu_{bp}, \quad \gamma_{bp} = 1 + a_{bp}(\gamma - 1)\end{aligned}\tag{348}$$

where  $a_{tf}$ ,  $a_{sp}$ , and  $a_{bp}$  are stress scaling parameters that can be specified in the `ThinLayerMobility` parameter set, and  $\gamma$  is an isotropic enhancement factor calculated from a stress-dependent `Factor` model (see [Isotropic Factor Models on page 988](#)).

## Auto-Orientation and Named Parameter Sets

The thin-layer mobility model implicitly supports the auto-orientation framework (see [Auto-Orientation Framework on page 86](#)) and named parameter sets (see [Named Parameter Sets on page 85](#)). That is, auto-orientation or named parameter sets for the `ThinLayerMobility` parameters will be used (and required) if these options are invoked for the normal field-dependent interface model used in conjunction with the thin-layer mobility model.

For example, this specification:

```
Physics {
    Mobility (
        PhuMob
        ThinLayer (Lombardi(AutoOrientation))
    )
}
```

invokes auto-orientation for both the Lombardi-specific calculations *and* the thin layer-specific calculations. In this case, Sentaurus Device requires orientation-dependent parameter sets for both the `ENormalDependence` parameters (for Lombardi) and the `ThinLayerMobility` parameters.

As another example, the following specification uses both the `IALMob "110"` parameter set and the `ThinLayerMobility "110"` parameter set:

```
Physics {
    Mobility (
        ThinLayer (IALMob(ParameterSetName="110"))
    )
}
```

## Chapter 15: Mobility

### Carbon Nanotube Mobility Model

## Geometric Parameters

The `ThinLayer` subcommand has the same `<geo_parameters>` as the `LayerThickness` command (see [Geometric Parameters of LayerThickness Command on page 381](#)).

---

## Carbon Nanotube Mobility Model

The carbon nanotube (CNT) mobility model is specifically for CNT applications. The model is based on the multivalley band structure (see [Multivalley Band Structure on page 327](#)) and, particularly, on the CNT DOS (see [Carbon Nanotube Density-of-States on page 331](#)). In low-driving fields and in the relaxation time approximation, the 1D conductance of one CNT band can be expressed using the Landauer approach and as shown in [20]:

$$G_i = \frac{4q^2}{\hbar} \sum_0^\infty \frac{\lambda_i(\varepsilon)}{L_{ch} + \lambda_i(\varepsilon)} - \frac{\partial f_0(\varepsilon)}{\partial \varepsilon} d\varepsilon \quad (349)$$

where  $G_i$  is the 1D conductance of the  $i$ -band,  $\lambda_i(\varepsilon)$  is the energy-dependent mean free path (MFP) of carriers in the  $i$ -band,  $L_{ch}$  is the CNT channel length, and  $f_0(\varepsilon)$  is the Fermi–Dirac distribution function.

With the known band conductance  $G_i$  from [Equation 349](#), a local averaged low-field mobility is computed as follows:

$$\mu = \frac{L_{ch}}{qA} \frac{G_i}{n_i} \quad (350)$$

where  $n_i$  is the local carrier concentration of the  $i$ -band, and  $A$  is the CNT-conducting area as used in [Equation 207](#) for CNT DOS.

Accounting for the optical (OP) and acoustic (AC) phonon scattering, both intraband and interband scattering, Pauli blocking for final states (all as proposed in [20]), and the MFP  $\lambda_i(\varepsilon)$  of carriers in the  $i$ -band (in [Equation 349](#)) are computed as follows:

$$\begin{aligned} \frac{1}{\lambda_i(\varepsilon)} &= \frac{1}{\lambda_{AP(i)}(\varepsilon)} + \frac{1-f_0(\varepsilon + \hbar\omega_{OP})}{\lambda_{OP, ik}^{abs}(\varepsilon)} + \frac{1-f_0(\varepsilon - \hbar\omega_{OP})}{\lambda_{OP, ik}^{ems}(\varepsilon)} \\ \lambda_{AP(i)}(\varepsilon) &= \lambda_{AP} \frac{300}{T} \frac{D_k(\infty)}{D_k(\varepsilon)} \frac{v_i(\varepsilon)}{v_i(\infty)} \\ \lambda_{OP, ik}^{ems/abs}(\varepsilon) &= \lambda_{OP} \frac{N_{OP}(300) + 1}{N_{OP}(T) + 1/2 \pm 1/2} \frac{D_k(\infty)}{D_k(\varepsilon \mp \hbar\omega_{OP})} \frac{v_i(\varepsilon)}{v_i(\infty)} \\ v_i(\varepsilon) &= v_F \sqrt{1 - \frac{(E_g/2 + \varepsilon_i)^2}{(E_g/2 + \varepsilon_i + \varepsilon)^2}} \quad . \\ N_{OP}(T) &= 1 / (\exp(\hbar\omega_{OP}/kT) - 1) \end{aligned} \quad (351)$$

## Chapter 15: Mobility

### Carbon Nanotube Mobility Model

where the sum  $\Sigma$  goes over all used CNT bands and, for  $i = k$ , the intraband scattering is accounted for. However, for all other cases, the interband scattering is accounted for by different user-defined MFP model parameters,  $\lambda_{AP}$  and  $\lambda_{OP}$ , set for the temperature  $T$  equal to 300 K. The upper (lower) signs in [Equation 351](#) correspond to the emission (absorption) of optical phonons.

The CNT DOS  $D_k(\varepsilon)$  is defined by [Equation 207](#) and, correspondingly,  $D_k(\infty) = 4/(\pi\hbar v_F A)$ . Based on such a CNT DOS,  $v_i(\varepsilon)$  is the energy-dependent carrier velocity and, correspondingly,  $v_i(\infty) = v_F$  is the Fermi velocity. The optical phonon energy  $\hbar\omega_{OP}$  is a user-defined parameter and  $N_{OP}(T)$  is the optical phonon occupation function.

---

## Using the CNT Mobility Model

To activate the CNT mobility model, use the following either globally or regionwise in the `Physics` section of the command file:

```
eMobility (
    CNTMob( Lch = <value> )    # <value> in [nm]
)
```

If the `CNTMob` model is used with other low-field mobility models defined in the `Mobility` section, then Matthiessen's rule is applied.

The model parameters used in [Equation 351](#) should be set in the following section of the parameter file:

```
CNTMobility (
    hbarOmega =      0.18,    0.18    # [eV],   $\hbar\omega_{OP}$ 
    Lambda_OP =     60,      60      # [nm],   $\lambda_{OP}$  in intraband scattering
    Lambda_AP =     1.12e3,  1.12e3 # [nm],   $\lambda_{AP}$  in intraband scattering
    Lambda_OP_inter = 60,      60      # [nm],   $\lambda_{OP}$  in interband scattering
    Lambda_AP_inter = 1.12e3,  1.12e3 # [nm],   $\lambda_{AP}$  in interband scattering
)
```

The optical phonon energy and MFP model parameters in the example correspond to those used in [\[20\]](#) for a CNT diameter equal to 4 nm.

### Note:

The CNT mobility model works only with CNT bands that are defined in the parameter file (with `dosmodel=CNT`; see [Using Multivalley Band Structure on page 331](#)). If CNT bands are not set, then Sentaurus Device stops the simulation.

## Chapter 15: Mobility

### Low-Temperature Phonon Exponent Correction for Mobility Models

The following example shows CNT band settings, which correspond to five bands used in [20] for a CNT diameter equal to 4 nm:

```
Multivalley (
    eValley"Band1" ( energy = 0      dosmodel = CNT )
    eValley"Band2" ( energy = 0.102  dosmodel = CNT )
    eValley"Band4" ( energy = 0.306  dosmodel = CNT )
    eValley"Band5" ( energy = 0.408  dosmodel = CNT )
    eValley"Band7" ( energy = 0.612  dosmodel = CNT )
)
```

To plot the CNT mobility from [Equation 350](#), use the `eCNTMobility` and `hCNTMobility` keywords in the `Plot` section for electrons and holes, respectively.

#### Note:

[Equation 349](#) involves a numeric computation of the integral over the energy for each CNT band. Such an integration is performed using quadratures as for the multivalley model (see [Using Multivalley Band Structure on page 331](#)). To improve the accuracy of the computation, you can use a larger number of energy points, for example, by setting `Math{ DensityIntegral(100) }`. This might be needed for low-temperature simulations where  $\frac{d\mathcal{J}_0(\epsilon)}{d\epsilon}$  is a function over the energy with a sharp peak at the carrier Fermi energy.

---

## Low-Temperature Phonon Exponent Correction for Mobility Models

For mobility models (constant mobility, Philips unified mobility, enhanced Lombardi mobility, and `IALMob`), the exponent of temperature ( $\theta$  or  $\xi$  for 3D phonons, and  $k$  for 2D phonons) in the various phonon mobility models ([Equation 251](#), [Equation 271](#), [Equation 285](#), [Equation 308](#), and [Equation 309](#)) is assumed to be constant with temperature. However, temperature-dependent phonon mobility measurement [21][22] shows that the exponent itself varies with temperature, particularly at low temperatures ( $T < 150$  K). This temperature-dependent correction can be incorporated by an empirically fitted formula (for both electrons and holes) for 3D phonons:

$$\theta(T) = \theta + \Delta\theta \tanh\left[\frac{T}{T_c}\right] - 1 \quad (352)$$

For 2D phonons, it is:

$$k(T) = k + \Delta\theta_{2D} \tanh\left[\frac{T}{T_{c,2D}}\right] - 1 \quad (353)$$

where  $\theta$  and  $k$  are existing constant values, and  $\Delta\theta$  ( $\Delta\theta_{2D}$ ) and  $T_c$  ( $T_{c,2D}$ ) are user inputs. By default, the value of  $\Delta\theta$  and  $\Delta\theta_{2D}$  is 0, and the correction is deactivated. The correction

## Chapter 15: Mobility

### Low-Temperature Phonon Exponent Correction for Mobility Models

can be switched on by defining nonzero values for  $\Delta\theta$  and  $\Delta\theta_{2D}$  in the parameter file for the particular mobility model that is being used in the simulation.

For example, when using the Philips unified mobility model (`PhuMob`) in the `Physics` section, the low-temperature correction for phonons (`PhuMob` has only a 3D phonon component, [Equation 269](#)) can be switched on by specifying the values of  $\Delta\theta$  and  $T_c$  for electron and holes separately:

```
PhuMob {  
    tc_e      = 150.0  
    tc_h      = 65.0  
    deltheta_e = 0.5  
    deltheta_h = 0.43  
}
```

The next sections provide default and recommended values for  $\Delta\theta$  ( $\Delta\theta_{2D}$ ) and  $T_c$  ( $T_{c,2D}$ ) for the various mobility models.

---

## Constant Mobility Model

This mobility model considers a 3D phonon model (see [Equation 251](#)). The recommended values for the low-temperature correction for silicon are given here.

*Table 69 Low-temperature correction: recommended values of ConstantMobility for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$\Delta\theta$	<code>deltheta</code>	0.5	0.43	1
$T_c$	<code>tc</code>	150.0	65.0	K

It can be activated by using these values in the parameter file. The default is  $\Delta\theta=0$ .

```
ConstantMobility:  
{  
    tc      = 150.0, 65.0    # [K]  
    deltheta = 0.5, 0.43     #[1]  
}
```

---

## Philips Unified Mobility Model

This mobility model considers a 3D phonon model (see [Equation 271](#)). The recommended values for the low-temperature correction for silicon are given here. Note that the electron values are the same for both arsenic and phosphorus doping, and the hole value is the same for boron doping.

## Chapter 15: Mobility

### Low-Temperature Phonon Exponent Correction for Mobility Models

Table 70 Low-temperature correction: recommended values of PhuMob for silicon

Symbol	Parameter name	Value	Unit
$\Delta\theta_e$	deltheta_e	0.5	1
$\Delta\theta_h$	deltheta_h	0.43	1
$T_{c,e}$	tc_e	150.0	K
$T_{c,h}$	tc_h	65.0	K

It can be activated by using these values in the parameter file. The default is  $\Delta\theta_{e,h}=0$ .

```
PhuMob
{
    tc_e = 150.0
    tc_h = 65.0
    deltheta_e = 0.5
    deltheta_h = 0.43
}
```

---

## Enhanced Lombardi Mobility Model

This mobility model considers a 2D phonon model as part of the surface contribution (see [Equation 285](#)). The change in the exponent for 2D phonon is not well established in the literature; however, a few measurements [22] show an increase in the exponent at extremely low temperatures (<10 K). Based on carrier-acoustic phonon scattering calculations for a perfect 2D carrier gas with a single band, there are recommended values for silicon, but you should use your judgment when using these values.

Table 71 Low-temperature correction: recommended values of Lombardi model for silicon

Symbol	Parameter name	Electrons	Holes	Unit
$\Delta\theta_{2D}$	deltheta	-1.25	-1.25	1
$T_{c,2D}$	tc	16.0	16.0	K

It can be activated by using these values in the `EnormalDependence` parameter file. The default is  $\Delta\theta_{e,h,2D}=0$ .

```
EnormalDependence {
    tc      = 16.0 , 16.0
    deltheta = -1.25, -1.25
}
```

---

## Inversion and Accumulation Layer Mobility Model

This mobility model combines both 3D phonon ([Equation 308](#)) and 2D phonon ([Equation 309](#)) components. The recommended values for the low-temperature correction parameters for 3D and 2D phonons are given here.

*Table 72 Low-temperature correction: recommended values of IALMob for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$\Delta\theta$	deltheta	0.5	0.43	1
$T_c$	tc	150.0	65.0	K
$\Delta\theta_{2D}$	deltheta_2d	-1.25	-1.25	1
$T_{c, 2D}$	tc_2d	16.0	16.0	K

Like the Lombardi model correction, the change in the exponent for 2D phonons is not well established in the literature, but a few measurements [22] show the exponent increases at extremely low temperatures (<10 K). Based on carrier-acoustic phonon scattering calculations for a perfect 2D carrier gas with a single band, the recommended values for silicon are provided, but you should use your judgment when using these values for 2D phonons.

The low-temperature correction can be activated for `IALMob` by defining these values in the parameter file.

```
IALMob:
{
    tc          = 150.0, 65.0
    deltheta    = 0.5,   0.43
    deltheta_2d = -1.25, -1.25
    tc_2d      = 16.0,  16.0
}
```

The default value of these parameters is 0.

---

## High-Field Saturation Models

In high electric fields, the carrier drift velocity is no longer proportional to the electric field, instead, the velocity saturates to a finite speed  $v_{sat}$ .

Sentaurus Device supports different models for the description of this effect:

- The Canali model, two transferred electron models, and two PMIs are available for all transport models.
- The basic model and the Meinerzhagen–Engl model both require hydrodynamic simulations.
- Another flexible model for hydrodynamic simulation is described in [Energy-Dependent Mobility on page 878](#).

---

## Using High-Field Saturation Models

The high-field saturation models comprise three submodels: the actual mobility model, the velocity saturation model, and the driving force model. With some restrictions, these models can be freely combined.

The actual mobility model is selected by flags to `eHighFieldSaturation` or `hHighFieldSaturation`. The default is the Canali model (see [Extended Canali Model on page 440](#)).

The flag `TransferredElectronEffect` selects the transferred electron model (see [Transferred Electron Model on page 442](#)).

Similarly, an alternative transferred electron model is activated by the flag `TransferredElectronEffect2` (see [Transferred Electron Model 2 on page 442](#)).

The flags `CarrierTempDriveBasic` and `CarrierTempDriveME` activate the ‘basic’ and the Meinerzhagen–Engl model, respectively (see [Basic Model on page 444](#) and [Meinerzhagen–Engl Model on page 445](#)). These two models require hydrodynamic simulations.

For the Canali model, the two transferred electron models, and the `PMI_HighFieldMobility` PMI, the driving force model is selected by a flag to `eHighFieldSaturation` or `hHighFieldSaturation`. Available flags are `GradQuasiFermi` (default), `Eparallel`, `EparallelToInterface`, and `CarrierTempDrive` (see [Driving Force Models on page 447](#)). The latter is only available in hydrodynamic simulations. The `PMI_HighFieldMobility2` PMI supports `EparallelToInterface`, `Eparallel`, and `ElectricField` for the electric-field driving force. For the ‘basic’ model and the Meinerzhagen–Engl model, the driving force is part of the actual mobility model and cannot be chosen independently.

For all except the ‘basic’ model and the PMIs, a velocity saturation model can be selected in the `HighFieldDependence` parameter set (see [Velocity Saturation Models on page 446](#)).

## Named Parameter Sets for the High-Field Saturation Models

The `HighFieldDependence` and `HydroHighFieldDependence` parameter sets can be named. For example, in the parameter file, you can write:

```
HighFieldDependence "myset" { ... }
```

to declare a parameter set with the name `myset`.

To use a named parameter set, specify its name with `ParameterSetName` as an option to `HighFieldSaturation`, `eHighFieldSaturation`, or `hHighFieldSaturation`. For example:

```
eMobility (
    HighFieldSaturation( ParameterSetName = "myset" ... ) ...
)
```

By default, the unnamed parameter set is used.

## Auto-Orientation for the High-Field Saturation Models

The `HighFieldSaturation` and `Diffusivity` models support the auto-orientation framework that switches between different `HighFieldDependence` named parameter sets based on the surface orientation of the nearest interface (see [Auto-Orientation Framework on page 86](#)).

To activate this feature, specify `AutoOrientation` as an option to either `Diffusivity` or `HighFieldSaturation` in the command file.

For example:

```
Mobility (
    HighFieldSaturation( AutoOrientation ... ) ...
)
```

## Extended Canali Model

The Canali model [23] originates from the Caughey–Thomas formula [24], but has temperature-dependent parameters, which were fitted up to 430 K by Canali *et al.* [23]:

$$\mu(F) = \frac{(\alpha + 1)\mu_{\text{low}}}{\alpha + \left[ 1 + \frac{(\alpha + 1)\mu_{\text{low}} F_{\text{hfs}}}{v_{\text{sat}}} \right]^{\beta}^{1/\beta}} \quad (354)$$

where  $\mu_{\text{low}}$  denotes the low-field mobility. Its definition depends on which of the previously described mobility models have been activated (see [Mobility due to Phonon Scattering on page 386](#) to [Philips Unified Mobility Model on page 398](#)).

## Chapter 15: Mobility

### High-Field Saturation Models

The exponent  $\beta$  is temperature dependent according to:

$$\beta = \beta_0 \frac{T}{300\text{ K}}^{\beta_{\text{exp}}} \quad (355)$$

Details about the saturation velocity  $v_{\text{sat}}$  and driving field  $F_{\text{hfs}}$  are discussed in [Velocity Saturation Models on page 446](#) and [Driving Force Models on page 447](#). All other parameters are accessible in the parameter set `HighFieldDependence`.

The silicon default values are listed in [Table 73 on page 441](#).

A modified version of the Canali model is the Hänsch model [14]. It is activated by setting the parameter  $\alpha = 1$ . The Hänsch model is part of the Lucent mobility model (see [Lucent Model on page 446](#)). When using the hydrodynamic driving force [Equation 369](#),  $\alpha$  must be zero.

For the hydrodynamic driving force, [Equation 369](#) can be substituted into [Equation 354](#). Solving for  $\mu$  yields the hydrodynamic Canali model:

$$\mu = \frac{\mu_{\text{low}}}{\left[ \sqrt{1 + \gamma^2 \max(w_c - w_0, 0)}^{\beta} + \gamma \max(w_c - w_0, 0)^{\beta/2} \right]^{2/\beta}} \quad (356)$$

$$\text{where } \gamma \text{ is given by } \gamma = \frac{1}{2} \frac{\mu_{\text{low}}}{q \tau_{e,c} v_{\text{sat}}}^{\beta/2}.$$

In this form, the model has a discontinuous derivative at  $w_0 = w_c$ , which can lead to numeric problems. Therefore, Sentaurus Device applies a smoothing algorithm in the carrier temperature region  $T < T_c < (1 + K_{dT})T$  to create a smooth transition between the low-field mobility  $\mu_{\text{low}}$  and the mobility given in [Equation 356](#).  $K_{dT}$  defaults to 0.2 and can be accessed in the parameter set `HighFieldDependence`.

*Table 73      Canali model parameters (default values for silicon)*

Symbol	Parameter name	Electrons	Holes	Unit
$\beta_0$	beta0	1.109	1.213	1
$\beta_{\text{exp}}$	betaexp	0.66	0.17	1
$\alpha$	alpha	0	0	1

## Transferred Electron Model

For GaAs and other materials with a similar band structure, a negative differential mobility can be observed for high driving fields. This effect is caused by a transfer of electrons into a energetically higher side valley with a much larger effective mass. Sentaurus Device includes a transferred electron model for the description of this effect, as given by [25]:

$$\mu = \frac{\mu_{\text{low}} + \frac{v_{\text{sat}} F_{\text{hfs}}^4}{E_0^4}}{1 + \frac{F_{\text{hfs}}^4}{E_0^4}} \quad (357)$$

Details of the saturation velocity  $v_{\text{sat}}$  and the driving field  $F_{\text{hfs}}$  are discussed in [Velocity Saturation Models on page 446](#) and [Driving Force Models on page 447](#). The reference field strength  $E_0$  can be set in the parameter set `HighFieldDependence`.

The `HighFieldDependence` parameter set also includes a variable  $K_{\text{smooth}}$ , which is equal to 1 by default. If  $K_{\text{smooth}} > 1$ , a smoothing algorithm is applied to the formula for mobility in the driving force interval  $F_{\text{vmax}} < F < K_{\text{smooth}} F_{\text{vmax}}$ , where  $F_{\text{vmax}}$  is the field strength at which the velocity is at its maximum,  $v_{\text{max}} = \mu F_{\text{vmax}}$ . In this interval, [Equation 357](#) is replaced by a polynomial that produces the same values and derivatives at the points  $F_{\text{vmax}}$  and  $K_{\text{smooth}} F_{\text{vmax}}$ . It is sometimes numerically advantageous to set  $K_{\text{smooth}} \approx 20$ .

*Table 74 Transferred electron model: Default parameters*

Symbol	Parameter name	Electrons	Holes	Unit
$E_0$	E0_TrEF	4000	4000	Vcm <sup>-1</sup>
$K_{\text{smooth}}$	Ksmooth_TrEF	1	1	1

## Transferred Electron Model 2

Sentaurus Device provides an alternative high-field saturation mobility model for III–nitride materials:

$$\mu = \frac{\mu_{\text{low}} + \mu_1 \frac{F_{\text{hfs}}^\alpha}{E_0^\alpha} + v_{\text{sat}} \frac{F_{\text{hfs}}^{\beta-1}}{E_1^\beta}}{1 + \gamma \frac{F_{\text{hfs}}^\alpha}{E_0^\alpha} + \frac{F_{\text{hfs}}}{E_1^\beta}} \quad (358)$$

## Chapter 15: Mobility

### High-Field Saturation Models

This model is a unification of the two models proposed in [26] and [27]. The model in [26] can be obtained by setting  $\mu_1 = 0$  and  $E_0 = E_1$ . Similarly, the model in [27] can be obtained by setting  $\gamma = 1$ .

The model parameters are specified in the `TransferredElectronEffect2` section of the parameter file:

```
TransferredElectronEffect2 (
    mul1 = 0, 0
    E0 = 220893.6, 4000
    E1 = 220893.6, 4000
    alpha = 0.7857, 0
    beta = 7.2044, 4
    gamma = 6.1973, 0
)
```

The default electron parameters for  $\text{Al}_x\text{Ga}_{1-x}\text{N}$  and  $\text{In}_x\text{Ga}_{1-x}\text{N}$  are taken from [26] and are shown in [Table 75](#) and [Table 76](#).

*Table 75 Transferred electron effect 2 model parameters in  $\text{Al}_x\text{Ga}_{1-x}\text{N}$*

Material	$\mu_1 \text{ [cm}^2\text{V}^{-1}\text{s}^{-1}]$	$E_0 = E_1 \text{ [Vcm}^{-1}]$	$\alpha$ [1]	$\beta$ [1]	$\gamma$ [1]
GaN	0	220893.6	0.7857	7.2044	6.1973
$\text{Al}_{0.2}\text{Ga}_{0.8}\text{N}$	0	245579.4	0.7897	7.8138	6.9502
$\text{Al}_{0.5}\text{Ga}_{0.5}\text{N}$	0	304554.1	0.8080	9.4438	8.0022
$\text{Al}_{0.8}\text{Ga}_{0.2}\text{N}$	0	386244.0	0.8324	12.5795	8.6037
AlN	0	447033.9	0.8554	17.3681	8.7253

*Table 76 Transferred electron effect 2 model parameters in  $\text{In}_x\text{Ga}_{1-x}\text{N}$*

Material	$\mu_1 \text{ [cm}^2\text{V}^{-1}\text{s}^{-1}]$	$E_0 = E_1 \text{ [Vcm}^{-1}]$	$\alpha$ [1]	$\beta$ [1]	$\gamma$ [1]
GaN	0	220893.6	0.7857	7.2044	6.1973
$\text{In}_{0.2}\text{Ga}_{0.8}\text{N}$	0	151887.0	0.7670	6.0373	5.1797
$\text{In}_{0.5}\text{Ga}_{0.5}\text{N}$	0	93815.1	0.7395	4.8807	3.7387
$\text{In}_{0.8}\text{Ga}_{0.2}\text{N}$	0	63430.5	0.6725	4.1330	2.7321
InN	0	52424.2	0.6078	3.8501	2.2623

For other materials, as well as for hole mobility, the parameters in [Table 77](#) are used.

*Table 77 Default transferred electron effect 2 model parameters*

$\mu_1 \text{ [cm}^2\text{V}^{-1}\text{s}^{-1}\text{]}$	$E_0 = E_1 \text{ [Vcm}^{-1}\text{]}$	$\alpha \text{ [1]}$	$\beta \text{ [1]}$	$\gamma \text{ [1]}$
0	4000	0	4	0

With these parameters, the model reverts to the standard transferred electron model (see [Transferred Electron Model on page 442](#)).

Sometimes, convergence problems are observed when the derivative of the velocity  $v = \mu F_{\text{hfs}}$  with respect to the driving force  $F_{\text{hfs}}$  becomes negative. As a potential solution, Sentaurus Device provides an option to specify a lower bound for this derivative:

```
Math {
    TransferredElectronEffect2_MinDerivativePerField = 0
}
```

If required, individual bounds can be specified for electrons and holes:

```
Math {
    TransferredElectronEffect2_eMinDerivativePerField = -1e-3
    TransferredElectronEffect2_hMinDerivativePerField = -1e-2
}
```

By default, Sentaurus Device applies the lower bound  $\partial v / \partial F_{\text{hfs}} \geq -10^{100} \text{ cm}^2 \text{V}^{-1} \text{s}^{-1}$ . Note that this lower bound cannot be applied to the hydrodynamic driving force (`CarrierTempDrive`).

## Multivalley Transferred Electron Mobility

This experimental model accounts for the transferred electron effect in III–V applications based on the usage of the Sentaurus Device hydrodynamic model (see [Hydrodynamic Model for Temperatures on page 252](#)) and the multivalley band-structure representation (see [Multivalley Band Structure on page 327](#)). For a description of the model and its settings, see [Multivalley Transferred Carrier Mobility Model on page 970](#).

## Basic Model

According to this very simple model, the mobility decays inversely with the carrier temperature:

$$\mu = \mu_{\text{low}} \frac{300 \text{ K}}{T_c} \quad (359)$$

where  $\mu_{\text{low}}$  is the low-field mobility and  $T_c$  is the carrier temperature.

---

## Meinerzhagen–Engl Model

According to the Meinerzhagen–Engl model [28], the high field mobility degradation is given by:

$$\mu = \frac{\mu_{\text{low}}}{\left[ 1 + \frac{\mu_{\text{low}}}{2q \tau_{e,c} v_{\text{sat}}^2} \frac{3k(T_c - T)}{2} \right]^{1/\beta}} \quad (360)$$

where  $\tau_{e,c}$  is the energy relaxation time.

The coefficients of the saturation velocity  $v_{\text{sat}}$  (see [Equation 361](#)) and the exponent  $\beta$  (see [Equation 355](#)) are accessible in the parameter file:

```
HighFieldDependence {
    vsat0    = <value for electrons> <value for holes>
    vsatexp = <value for electrons> <value for holes>
}
HydroHighFieldDependence {
    beta0    = <value for electrons> <value for holes>
    betaexp = <value for electrons> <value for holes>
}
```

The silicon default values are given in [Table 78](#) and [Table 79](#).

*Table 78 Meinerzhagen–Engl model: Default parameters*

Silicon	Electrons	Holes	Unit
$\beta_0$	0.6	0.6	1
$\beta_{\text{exp}}$	0.01	0.01	1

---

## Physical Model Interfaces

Sentaurus Device offers different physical model interfaces (PMIs) for high-field mobility saturation:

- The first PMI is activated by specifying the name of the model as an option of `HighFieldSaturation`, `eHighFieldSaturation`, or `hHighFieldSaturation`. See [High-Field Saturation on page 1287](#).

## Chapter 15: Mobility

### High-Field Saturation Models

- The second PMI allows you to implement models that depend on two driving forces. It is specified by `PMIModel` as the option to `HighFieldSaturation`, `eHighFieldSaturation`, or `hHighFieldSaturation`. See [High-Field Saturation With Two Driving Forces on page 1295](#).

---

## Lucent Model

The Lucent model has been developed by Darwish *et al.* [12]. Sentaurus Device implements this model as a combination of:

- An extended Philips unified mobility model (see [Philips Unified Mobility Model on page 398](#)) with the parameters  $f_e = 0$  and  $f_h = 0$  (see [Table 59 on page 402](#)). The Lucent model described in [12] also does not include clustering. To disable clustering in the Philips unified mobility model, set the parameters  $N_{\text{ref},A}$  and  $N_{\text{ref},D}$  to very large numbers.
- The enhanced Lombardi model (see [Enhanced Lombardi Model on page 404](#)) with the parameters from [Table 61 on page 406](#).
- The Hänsch model (see [Extended Canali Model on page 440](#)) with the parameter  $\alpha = 1$  (see [Table 73 on page 441](#)). In addition, the  $\beta$  exponent in the Hänsch model is equal to 2, which can be accomplished by setting  $\beta_0 = 2$  and  $\beta_{\text{exp}} = 0$ .

---

## Velocity Saturation Models

Sentaurus Device supports different velocity saturation models.

Model 1 is part of the Canali model and is given by:

$$v_{\text{sat}} = v_{\text{sat},0} \left( \frac{300 \text{ K}}{T} \right)^{\frac{v_{\text{sat},\text{exp}}}{2}} \quad (361)$$

This model is recommended for silicon.

Model 2 is recommended for GaAs. Here,  $v_{\text{sat}}$  is given by:

$$v_{\text{sat}} = \begin{cases} A_{\text{vsat}} - B_{\text{vsat}} \left( \frac{T}{300 \text{ K}} \right)^{\frac{v_{\text{sat}}}{2}} & v_{\text{vsat}} > v_{\text{sat,min}} \\ v_{\text{sat,min}} & \text{otherwise} \end{cases} \quad (362)$$

Parameters of both models are accessible in the `HighFieldDependence` parameter set.

## Selecting Velocity Saturation Models

The variable `vsat_formula` in the `HighFieldDependence` parameter set selects the velocity saturation model. If `vsat_formula` is set to 1, [Equation 361](#) is used. If

`Vsat_formula` is set to 2, then [Equation 362](#) is selected. The default value of `Vsat_formula` depends on the semiconductor material, for example, for silicon the default is 1; for GaAs, it is 2.

*Table 79 Velocity saturation parameters*

Symbol	Parameter name	Electrons	Holes	Unit
$v_{\text{sat},0}$	<code>vsat0</code>	$1.07 \times 10^7$	$8.37 \times 10^6$	cm/s
$v_{\text{sat,exp}}$	<code>vsatexp</code>	0.87	0.52	1
$A_{\text{vsat}}$	<code>A_vsat</code>	$1.07 \times 10^7$	$8.37 \times 10^6$	cm/s
$B_{\text{vsat}}$	<code>B_vsat</code>	$3.6 \times 10^6$	$3.6 \times 10^6$	cm/s
$v_{\text{sat,min}}$	<code>vsat_min</code>	$5.0 \times 10^5$	$5.0 \times 10^5$	cm/s

## Driving Force Models

Sentaurus Device supports five models for the driving force  $F_{\text{hfs}}$ , the first two of which also are affected by the `ParallelToInterfaceInBoundaryLayer` keyword (see [Field Correction Close to Interfaces on page 452](#)). For a summary of keywords, see [Table 279 on page 1666](#).

## Electric Field Parallel to the Current

For the first model (flag `Eparallel`), the driving field for electrons is the electric field parallel to the electron current density:

$$F_{\text{hfs},n} = \vec{F} \cdot \hat{\vec{J}_n} \quad (363)$$

To avoid numeric noise in the calculation of the `Eparallel` driving force, Sentaurus Device provides an option of setting  $F_{\text{hfs},n}$  to zero for very small current densities, defined as:

$$\frac{J_n}{\mu_n} < \frac{c_{\text{min}}}{\mu_{\text{ref}}} \quad (364)$$

Here,  $\mu_{\text{ref}} = 1000 \text{ cm}^2 \text{V}^{-1} \text{s}^{-1}$  is an arbitrary reference mobility, and  $c_{\text{min}}$  is specified (in  $\text{Acm}^{-2}$ ) by `CDensityMin` in the `Math` section (default is 0).

The driving field for holes is analogous.

The electric field parallel to the current is the physically correct driving force for high-field saturation mobility models as well as for avalanche generation. Unfortunately, this driving force suffers from numeric instabilities for small currents because the direction of the current is not well defined and fluctuates easily. Therefore, the radius of convergence can be very small. The following driving forces are provided as alternatives with improved numeric stability.

## Gradient of Quasi-Fermi Potential

For the second model (flag `GradQuasiFermi`), the driving field for electrons is:

$$F_{\text{hfs}, n} = |\vec{\nabla}\Phi_n| \quad (365)$$

By default, the electric field replaces the gradient of the quasi-Fermi potential within mesh elements touching a contact:

$$F_{\text{hfs}, n} = |\vec{F}| \quad (366)$$

In the `Math` section, you can request that [Equation 365](#) is used for all elements:

```
Math {
    ComputeGradQuasiFermiAtContacts = UseQuasiFermi
}
```

The driving field for holes is analogous.

**Note:**

Usually, [Equation 363](#) and [Equation 365](#) give the same or very similar results. However, numerically, one model might prove to be more stable. For example, in regions with small current, the evaluation of the parallel electric field can be numerically problematic.

This is the default driving force for drift-diffusion simulations.

## Electric Field Parallel to the Interface

The third model (keyword `EparallelToInterface`) computes the driving force as the electric field parallel to the closest semiconductor–insulator interface:

$$F_{\text{hfs}} = \left| (\hat{I} - \hat{n}\hat{n}^T)\vec{F} \right| \quad (367)$$

The vector  $\hat{n}$  is a unit vector pointing to the closest semiconductor–insulator interface. It is determined in the same way as for the mobility degradation at interfaces. To select explicitly a semiconductor–insulator interface, use the `EnormalInterface` specification in the `Math` section (see [Normal to the Interface on page 426](#)).

The driving force of `EparallelToInterface` is the same for electrons and holes. It is numerically stable because it does not depend on the direction of the current. However, this

## Chapter 15: Mobility

### High-Field Saturation Models

model will only give valid results if the current flows predominantly parallel to the interface (such as in the channel of MOSFET devices).

In certain situations, for example, in the channel of a FinFET, the direction of the current is known. In this case, you can specify a constant direction vector  $\vec{d}$  in the `Math` section:

```
Math {
    EparallelToInterface (
        Direction = (1 0 0)
    )
}
```

Then, the driving force is computed as the electric field  $\vec{F}$  parallel to the direction vector  $\vec{d}$ :

$$F_{\text{hfs}} = \frac{\max(\vec{d} \cdot \vec{F}, 0)}{|\vec{d}|} \quad (368)$$

#### Note:

[Equation 368](#) ensures a nonnegative driving force  $F_{\text{hfs},n}$ . If the scalar product between the direction vector  $\vec{d}$  and the electric field  $\vec{F}$  becomes negative, the driving force will be set to zero. This effectively switches off high-field saturation.

Therefore, it is crucial that the direction of vector  $\vec{d}$  is aligned with the direction of the current flow. Otherwise, you might inadvertently disable the high-field saturation mobility model.

An `EparallelToInterface` specification can appear in the global `Math` section, as well as in materialwise or regionwise `Math` sections. If no direction vector, or a zero direction vector, has been specified, the driving force will revert to [Equation 367](#).

It also might be required to restrict the validity of the direction vector  $\vec{d}$  to only a part of a device. This can be accomplished by specifying a list of boxes together with the required direction vector:

```
Math {
    EparallelToInterface (      # 2D example
        Direction = (1 0)
        Box = ((1 1) (3 4))
        Box = ((3 3) (5 4))
    )

    EparallelToInterface (      # 3D example
        Direction = (0 0 1)
        Box = ((1 1 0) (3 4 1))
        Box = ((3 3 0) (5 4 1))
    )
}
```

Boxes are specified by the coordinates of the corners of a diagonal (units of  $\mu\text{m}$ ). The driving force in [Equation 368](#) is then only applied to mesh vertices that are contained in at least one

of the boxes. On all other vertices, the driving force in [Equation 367](#) is used. You can specify different direction vectors  $\vec{d}$  in different parts of a device by using multiple `EparallelToInterface` statements in the `Math` section.

You also can plot the normalized direction vector  $\vec{d}/|\vec{d}|$  in the `Plot` section:

```
Plot {
    EP2I_Direction/Vector
}
```

This can be useful to visualize the areas where the driving force in [Equation 368](#) applies.

## Hydrodynamic Driving Force

The fourth model (keyword `CarrierTempDrive`) requires hydrodynamic simulation. The driving field for electrons is:

$$F_{\text{hfs}, n} = \sqrt{\frac{\max(w_n - w_0, 0)}{\tau_{e, n} q \mu_n}} \quad (369)$$

where  $w_n = 3kT_n/2$  is the average electron thermal energy,  $w_0 = 3kT/2$  is the equilibrium thermal energy, and  $\tau_{e, n}$  is the energy relaxation time. The driving fields for holes are analogous.

This is the default driving force for hydrodynamic simulations.

## Electric Field

The fifth model (keyword `ElectricField`) uses the electric field as an approximation for  $F_{\text{hfs}}$ .

## Interpolation of Driving Forces to Zero Field

For numeric reasons, Sentaurus Device actually implements the following generalizations of [Equation 363](#) and [Equation 365](#):

$$F_{\text{hfs}, n} = \frac{n}{n + n_0} \vec{F} \cdot \hat{\vec{J}_n} \quad (370)$$

$$F_{\text{hfs}, n} = \frac{n}{n + n_0} |\nabla \Phi_n| \quad (371)$$

Here,  $n_0$  is a numeric damping parameter. The values for electrons and holes default to zero, and are set (in  $\text{cm}^{-3}$ ) in the `Math` section with the `RefDens_eGradQuasiFermi_Zero` and `RefDens_hGradQuasiFermi_Zero` parameters. Using positive values for  $n_0$  can improve convergence for problems where strong generation–recombination occurs in regions with small density.

Instead of `RefDens_eGradQuasiFermi_Zero` and `RefDens_hGradQuasiFermi_Zero`, the old aliases `eDrForceRefDens` and `hDrForceRefDens` can be used as well.

## Interpolation of the GradQuasiFermi Driving Force

Occasionally, convergence problems can be attributed to the `GradQuasiFermi` driving force, particularly when  $\nabla\Phi_n$  changes rapidly for small changes in the electron density  $n$ . In such cases, you can use the gradient of a modified quasi-Fermi potential  $\tilde{\Phi}_n$  instead. In the case of Boltzmann statistics, the modified quasi-Fermi potentials are given by:

$$\tilde{\Phi}_n = \phi - \phi_{\text{ref}} + \frac{\chi}{q} - \frac{kT}{q} \log \frac{n + n_0}{N_C} \quad (372)$$

$$\tilde{\Phi}_p = \phi - \phi_{\text{ref}} + \frac{\chi}{q} + \frac{E_{g,\text{eff}}}{q} + \frac{kT}{q} \log \frac{p + p_0}{N_V} \quad (373)$$

The equivalent expressions for Fermi statistics are:

$$\tilde{\Phi}_n = \phi - \phi_{\text{ref}} + \frac{\chi}{q} - \frac{kT}{q} F_{1/2}^{-1} \left[ \frac{n + n_0}{N_C} \right] \quad (374)$$

$$\tilde{\Phi}_p = \phi - \phi_{\text{ref}} + \frac{\chi}{q} + \frac{E_{g,\text{eff}}}{q} + \frac{kT}{q} F_{1/2}^{-1} \left[ \frac{p + p_0}{N_V} \right] \quad (375)$$

The values of  $n_0$  and  $p_0$  can be specified with the following parameters in the `Math` section:

- `RefDens_eGradQuasiFermi_ElectricField_HFS`
- `RefDens_hGradQuasiFermi_ElectricField_HFS`

For  $n_0 = 0$ , you have  $\nabla\tilde{\Phi}_n = \nabla\Phi_n$  and, for  $n_0 \rightarrow \infty$ , you have  $\nabla\tilde{\Phi}_n \rightarrow \vec{F}$  as a limit (in the isothermal case). Therefore, this approach represents an interpolation between the gradient of the quasi-Fermi potential  $\nabla\Phi_n$  and the electric field  $\vec{F}$ .

These interpolation parameters affect only the computation of the `GradQuasiFermi` driving force for high-field saturation mobility. If you want the driving force for both high-field saturation mobility *and* avalanche generation to use interpolation to the electric field, you can use the following keywords (see [Interpolation of Avalanche Driving Forces on page 509](#)):

- `RefDens_eGradQuasiFermi_ElectricField`
- `RefDens_hGradQuasiFermi_ElectricField`

As an alternative, Sentaurus Device also provides an interpolation between the gradient of the quasi-Fermi potential and the electric field parallel to the interface:

$$F_{\text{hfs},n} = \frac{n}{n + n_0} |\nabla\Phi_n| + \frac{n_0}{n + n_0} \left| (I - \hat{n}\hat{n}^T) \vec{F} \right| \quad (376)$$

The reference densities for electrons and holes can be specified in the `Math` section by the following parameters:

- `RefDens_eGradQuasiFermi_EparallelToInterface_HFS`
- `RefDens_hGradQuasiFermi_EparallelToInterface_HFS`

## Interpolation of the Eparallel Driving Force

As with the `GradQuasiFermi` driving force, convergence problems associated with the `Eparallel` driving force can sometimes be alleviated with an interpolation to the electric field  $F$  at low carrier concentrations:

$$F_{\text{hfs}, n} = \frac{n}{n + n_0} \vec{F} \cdot \hat{\vec{J}_n} + \frac{n_0}{n + n_0} \left| \vec{F} \right| \neq$$
 (377)

In this case, the reference densities for electrons and holes can be specified in the `Math` section by the following parameters:

- `RefDens_eEparallel_ElectricField_HFS`
- `RefDens_hEparallel_ElectricField_HFS`

Interpolation of the `Eparallel` driving force to the electric field for avalanche generation can be specified with separate parameters (see [Interpolation of Avalanche Driving Forces on page 509](#)).

## Field Correction Close to Interfaces

`ParallelToInterfaceInBoundaryLayer` in the `Math` section controls the computation of driving forces for mobility and avalanche models along interfaces. With this switch, the avalanche and mobility computations in boundary elements along interfaces use only the component parallel to the interface of the following vectors:

- Current vector
- Gradient of the quasi-Fermi potential

In this context, an interface is either a semiconductor–insulator region interface or an external boundary interface of the device.

This switch can be useful to avoid nonphysical breakdowns along an interface with a coarse mesh. It can be specified regionwise, in which case, it applies only to boundary elements in a given region.

The switch `ParallelToInterfaceInBoundaryLayer` supports two “layer” options:

```
Math {
    ParallelToInterfaceInBoundaryLayer (PartialLayer)
    ParallelToInterfaceInBoundaryLayer (FullLayer)
}
```

If `PartialLayer` is specified, parallel fields are only used in elements that are connected to the interface by an edge (in 2D) or a face (in 3D). This is the default. The option `FullLayer` uses parallel fields in all elements that touch the interface by either a face, an edge, or a vertex.

The switch `ParallelToInterfaceInBoundaryLayer` is activated by default. It can be deactivated by specifying `-ParallelToInterfaceInBoundaryLayer`. Additional options are available for `ParallelToInterfaceInBoundaryLayer` to deactivate it only on portions of the interface. Specifying `-ExternalBoundary` deactivates it on all external boundaries; whereas, `-ExternalXPlane`, `-ExternalYPlane`, and `-ExternalZPlane` deactivates it only on external boundaries perpendicular to the x-axis, y-axis, and z-axis, respectively. Specifying `-Interface` deactivates it on semiconductor-insulator region interfaces.

---

## Non-Einstein Diffusivity

By default, in the drift-diffusion equation (see [Drift-Diffusion Model on page 239](#)), Sentaurus Device assumes that the Einstein relation holds, and the diffusivity is related to the mobility by  $D_n = kT\mu_n$  and  $D_p = kT\mu_p$ . For short-channel devices with steep doping gradients, this relation is no longer valid. Therefore, Sentaurus Device allows you to compute the diffusivities independently from the mobilities.

To be able to reuse existing mobility models, Sentaurus Device expresses the diffusivities in terms of *diffusivity mobilities*,  $D_n = kT\mu_{n,\text{diff}}$  and  $D_p = kT\mu_{p,\text{diff}}$ , and you can specify models and parameters for  $\mu_{n,\text{diff}}$  and  $\mu_{p,\text{diff}}$ , which are different from  $\mu_n$  and  $\mu_p$ .

To compute the diffusivity mobilities, specify the keyword `Diffusivity`, `eDiffusivity`, or `hDiffusivity` as an option to `eMobility`, `hMobility`, or `Mobility`. The options for `eDiffusivity` and `hDiffusivity` are the same as for `HighFieldSaturation`. The low-field mobility that enters the diffusivity mobility is the same as for the high-field mobility.

The simplest way to obtain diffusivity mobilities different from the mobilities is to use named parameters sets to achieve a different parameterization (see [Named Parameter Sets for the High-Field Saturation Models on page 440](#)). For example:

```
eMobility (
    HighFieldSaturation(ParameterSetName = "mymobpara")
    Diffusivity(ParameterSetName = "mydiffpara")
    DopingDependence      * applies to both diffusivity and mobility
)
```

## Chapter 15: Mobility

### High-Field Saturation Models

It is also possible to use entirely different models for high-field mobilities and diffusivity mobilities.

Diffusivity mobilities can be plotted. The names of the datasets are `eDiffusivityMobility` and `hDiffusivityMobility`.

The implementation of non-Einstein diffusivities has several restrictions. In particular, the following models use a current density that assumes the Einstein relation still holds:

- The models to compute the driving fields for mobility ([Driving Force Models on page 447](#)) and avalanche generation (see [Driving Force on page 509](#))
- The J-model for trap cross-sections (see [J-Model for Cross Sections on page 556](#))
- The current densities that appear in [Equation 79](#) and [Equation 80](#)

Transport in magnetic fields is not supported. Support for anisotropy is restricted to the tensor grid method with current-independent anisotropy. The hydrodynamic model is supported, but it is not recommended for use with non-Einstein diffusivity.

---

## Band Tail Mobility

For some applications, it might be useful to separate high-field mobility for carriers in the main bands and for carriers in occupied band tails. To do this, you should set the band tail model (see [Band Tails on page 325](#)). It might be considered for applications where leakages occur through defect bands. Such bands could be represented by the `BandTailDOS(Gaussian)` model and, for example, you could use the `VRHMob` mobility model (see [Variable Range Hopping Transport Mobility on page 456](#)) to model such leakages through the defect bands.

With  $\mu_{bt}$  as the band tail mobility and  $\mu_B$  as the high-field saturation mobility of carriers in the conduction or valence bands, the total mobility for the continuity equation is computed as follows:

$$\mu = \mu_{bt} \frac{n_{bt}}{n} + \mu_B \frac{n_B}{n} \quad (378)$$

where  $n_{bt}/n$  is the carrier occupation of the band tails,  $n_B/n$  is the carrier occupation of the conduction or valence bands, and  $n = n_B + n_{bt}$  is the total carrier concentration that is the solution of the continuity equation.

To activate the use of the band tail mobility model, the following syntax can be considered for a global or regionwise `Physics` section:

```
Physics {
    eBandTailDOS(Gaussian)
    Mobility (
        HighFieldSaturation(GradQuasiFermi)
        BandTailHighFieldSaturation(VRHMob)
```

## Chapter 15: Mobility

### High-Field Mobility in Disordered Materials

```
    )  
}
```

Band tail mobilities  $\mu_{bt}$  in [Equation 378](#) can be plotted, where the names of the datasets are `eBandTailMobility` and `hBandTailMobility`. To plot the carrier occupation of the band tails  $n_{bt}/n$ , you should use the `eBandTailOccupation` and `hBandTailOccupation` dataset names for electrons and holes, respectively. To plot the carrier occupation of the conduction or valence bands  $n_B/n$ , you should use the `eAllValleyOccupation` and `hAllValleyOccupation` dataset names.

---

## High-Field Saturation Mobility Scaling

Sentaurus Device allows the mobility calculated by the `HighFieldSaturation` models to be scaled. The parameters `ku` and `kv` are used to scale low-field mobility ( $\mu_{low}$ ) and saturation velocity ( $v_{sat}$ ), respectively, which appear in the formulas presented in this section:

$$\mu_{low} \rightarrow ku \cdot \mu_{low} \quad (379)$$

$$v_{sat} \rightarrow kv \cdot v_{sat} \quad (380)$$

The parameters `ku` and `kv` are specified in the `HighFieldDependence` parameter set in the parameter file and have default values of 1.0.

---

## High-Field Mobility in Disordered Materials

Disordered materials (such as organic materials or other solids with many defects) can have a specific dependence on the mobility of the electric field along the transport direction without saturation of the carrier velocity. This section presents the available mobility models.

---

### Poole–Frenkel Mobility

Most organic semiconductors have mobilities dependent on the electric field. Sentaurus Device supports mobilities having a square-root dependence on the electric field, which is a typical mobility dependence for organic semiconductors. Mobility as a function of the electric field is given by:

$$\mu = \mu_0 \exp \left( -\frac{E_0}{kT} \exp \sqrt{F} \frac{\beta}{T} - \gamma \right) \quad (381)$$

where  $\mu_0$  is the low-field mobility,  $\beta$  and  $\gamma$  are fitting parameters,  $E_0$  is the effective activation energy, and  $F$  is the driving force (electric field).

## Chapter 15: Mobility

### High-Field Mobility in Disordered Materials

The parameters  $E_0$ ,  $\beta$ , and  $\gamma$  can be adjusted in the `PFMOb` section of the parameter file:

```
Material = "pentacene" {
...
  PFMob {
    beta_e = 1.1
    beta_h = 1.1
    E0_e = 0.0
    E0_h = 0.0
    gamma_e = 0.1
    gamma_h = 0.2
  }
...
}
```

with their default parameters  $\text{beta\_e}=\text{beta\_h}=0.1$ ,  $\text{E0\_e}=\text{E0\_h}=0$  (in eV), and  $\text{gamma\_e}=\text{gamma\_h}=0.0$ .

Since the derivative of the mobility with respect to the driving force is infinite at zero fields, the evaluation of this derivative is performed by replacing the square root of the driving force with an equivalent approximation having a finite derivative for the zero field. The approximation of the square root with the regular square root can be adjusted by specifying the parameter `delta_sqrtReg` in the `PFMOb` section of the parameter file. Typical values are in the range 0.1–0.0001; its default is 0.1.

The model can be activated for both electrons and holes by specifying the keyword `PFMOb` as a model for `HighFieldSaturation` in the `Mobility` section and the driving force as a parameter:

```
Physics(Region="pentacene") {
...
  Mobility (
    HighFieldSaturation(PFMob Eparallel)
  )
...
}
```

The model can also be selected for electrons or holes separately:

```
Physics { Mobility ([eHighFieldSaturation(PFMob Eparallel)]
                     [hHighFieldSaturation(PFMob Eparallel)]) ... }
```

#### Note:

Because the Poole–Frenkel mobility model is implemented through the high-field saturation framework, all the parameters specified for driving forces also apply to this model (see [Driving Force Models on page 447](#)).

---

## Variable Range Hopping Transport Mobility

The variable range hopping (VRH) mobility model was developed [\[29\]](#) to account for transport in disordered and organic solids where the carrier density could be described by a

## Chapter 15: Mobility

### High-Field Mobility in Disordered Materials

Gaussian DOS (see [Gaussian Density-of-States for Organic Semiconductors on page 322](#) and [Band Tails on page 325](#)).

This development suggests the VRH mobility model in the following general form:

$$\mu = \frac{v_0 b}{F} \left[ 1 + \frac{2 \exp \left[ \frac{\sigma^2}{(kT)^2} \right]}{\exp \left[ \frac{qbF}{kT} \right] - 1} \right] \quad (382)$$

where:

- $v_0$  is the hopping frequency.
- $b$  is the distance between hopping sites.
- $\sigma$  is the characteristic energy of the Gaussian DOS.
- $F$  is the driving field.
- $T$  is the temperature.
- $k$  is the Boltzmann constant.
- $q$  is the electron charge.

It also suggests a simplified form for the VRH mobility, which is not reduced at a high driving field  $F$ :

$$\mu = \frac{v_0 b}{2F} \exp \left[ -\frac{\sigma^2}{(kT)^2} \right] \left[ \exp \left[ \frac{qbF}{kT} \right] - 1 \right] \quad (383)$$

Both [Equation 382](#) and [Equation 383](#) are available to use in the following section of the parameter file where all of the model parameters previously mentioned can be set:

```
VRHMob {
    formula_e = 2
    v0_e      = 1e13    # [s^-1]
    b_e       = 3.6e-8 # [cm]
    sigma_e   = 0.05   # [eV]
    formula_h = 2
    v0_h      = 1e13    # [s^-1]
    b_h       = 3.6e-8 # [cm]
    sigma_h   = 0.05   # [eV]
}
```

where `formula_e|h=1` selects [Equation 383](#) and `formula_e|h=2` selects [Equation 382](#).

## Chapter 15: Mobility

### Ballistic Mobility Model

The model can be activated for both electrons and holes by specifying the keyword VRHMob as a model for HighFieldSaturation in the Mobility section of a global, regionwise, or materialwise Physics section:

```
Physics {
    ...
    Mobility (
        HighFieldSaturation(VRHMob Eparallel)
    )
    ...
}
```

Similar to the Poole–Frenkel model, all driving forces are also available for this model in the HighFieldSaturation statement (see [Driving Force Models on page 447](#)).

---

## Ballistic Mobility Model

To account for the ballistic effect in drift-diffusion transport, a ballistic mobility model can be combined with the total drift-diffusion mobility using Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{dd}} + \frac{1}{\mu_{bal}} \quad (384)$$

where  $\mu_{bal}$  represents the ballistic mobility. Sentaurus Device provides two models for  $\mu_{bal}$  that are described in the following sections:

- [Channel Length–Dependent Model](#)
- [Kinetic Velocity Model](#)

---

## Channel Length–Dependent Model

The simplest channel length–dependent ballistic mobility model [6] is given by the following expression:

$$\mu_{bal}(T) = k(T) \cdot L_{ch} \quad (385)$$

where  $L_{ch}$  is a user-defined parameter. In most cases, you will specify a value for  $L_{ch}$  (in nm) that corresponds to the actual channel length for the device being simulated.

According to [6],  $k(T)$  is expressed as:

$$k(T) = \frac{qv_T}{2k_B T} \quad v_T = \sqrt{\frac{2k_B T}{\pi m^*}} \quad (386)$$

where  $q$  is the electron charge,  $T$  is the lattice temperature,  $k_B$  is the Boltzmann constant, and  $v_T$  is the thermal velocity with the effective mass  $m^*$  in the transport direction.

## Chapter 15: Mobility

### Ballistic Mobility Model

If you define the parameter  $k_{300} = k(300 \text{ K})$ , then the effective mass  $m^*$  from [Equation 386](#) can be expressed as:

$$m^* = \frac{q^2}{(600 \text{ K})\pi k_B k_{300}^2} \quad (387)$$

Using  $k_{300}$ , [Equation 385](#) can be rewritten as:

$$\mu_{\text{bal}}(T) = k_{300} \sqrt{\frac{300 \text{ K}}{T}} \cdot L_{\text{ch}} \quad (388)$$

where  $k_{300}$  is a user-defined parameter specified in the parameter file. This parameter can be adjusted to match measured device characteristics.

## Kinetic Velocity Model

To avoid using the explicit channel length dependency in [Equation 388](#), the authors of [30] developed a kinetic velocity model (KVM) with the following terms: thermionic emission and free carrier acceleration. The thermionic emission term in the ballistic mobility is given by:

$$\mu_{\text{bal}}^T = \frac{\sqrt{\frac{k_B T}{m^*}} y_T \frac{q \psi}{k_B T}}{|\nabla \psi|} \quad y_T(x) = e^x \sqrt{\alpha_0 + 2(1 - e^{-x})} \quad (389)$$

where  $\psi$  is the quasi-Fermi potential with reference to the source contact and with a negative sign for holes,  $m^*$  is from [Equation 387](#) and can be adjusted using the parameter  $k_{300}$ , and  $\alpha_0$  is a factor to an initial kinetic velocity near source.

The free carrier acceleration term in the ballistic mobility is given by:

$$\mu_{\text{bal}}^B = \frac{\sqrt{\frac{k_B T}{m^*}} y_B \frac{q \psi}{k_B T}}{|\nabla \psi|} \quad y_B(x) = \sqrt{\alpha_0 + 2x} \quad (390)$$

The total KVM ballistic mobility is expressed as follows:

$$\frac{1}{\mu_{\text{bal}}} = \frac{\alpha_T}{\mu_{\text{bal}}^T} + \frac{\alpha_B}{\mu_{\text{bal}}^B} \quad (391)$$

where  $\alpha_T$  is the thermionic emission constant, which according to the theory of [31] should equal 1.52, and  $\alpha_B$  is the distribution function factor of the free carrier acceleration term, which should be effective mass (semiconductor material) dependent.

**Note:**

The free carrier acceleration term ([Equation 390](#)) is required if pure ballistic transport should be modeled using the drift-diffusion model. However, for typical MOSFET applications where carrier mobility is affected by carrier scattering in the channel, the thermionic emission term alone ([Equation 389](#) and [Equation 391](#)) should be sufficient.

## Fermi–Dirac Statistics

Both [Equation 388](#) and [Equation 391](#) are written for the Boltzmann statistics, but for the strong inversion regime in MOSFET applications, accounting for Fermi–Dirac statistics might be essential (especially for III–V materials). To account for that, the following correction is suggested in [30]:

$$\mu_{\text{bal}} = \tilde{\mu}_{\text{bal}} \sqrt{\frac{k_B T}{q} \frac{\partial \ln n}{\partial \psi}} \quad (392)$$

where  $\tilde{\mu}_{\text{bal}}$  is from [Equation 388](#) or [Equation 391](#), and  $n$  is the carrier concentration.

## Frensley Rule

Typically, the ballistic mobility is combined with the total drift-diffusion mobility  $\mu_{\text{dd}}$  using Matthiessen's rule ([Equation 384](#)), but [30] and [31] suggest the following Frensley rule to compute the final mobility  $\mu$ :

$$\frac{\mu}{\mu_{\text{dd}}} + \left( \frac{\mu}{\mu_{\text{bal}}} \right)^2 \leq 1 \quad (393)$$

## Using the Ballistic Mobility Model

To activate the channel length–dependent ballistic mobility ([Equation 388](#)) with available options, you can specify the following in the `Physics` section of the command file:

```
Mobility ( BalMob(Lch = 5.0 Fermi Frensley TempDep) )
```

Here, `Lch` is the channel length (in nm), the `Fermi` option activates the Fermi–Dirac correction ([Equation 392](#)), the `Frensley` option activates [Equation 393](#) for the final mobility  $\mu$  (if not specified, Matthiessen's rule is used), and the `TempDep` option activates the temperature dependency in [Equation 388](#). If `Lch` is not specified, the default value of  $10^7$  nm is used, which effectively disables the model.

The model has one user-defined fitting parameter  $k_{300}$  that is accessible in the `BalMob` parameter set in the parameter file.

## Chapter 15: Mobility

### Ballistic Mobility Model

Table 80      *BalMob model: Default coefficients for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$k_{300}$	k	20.0	20.0	$\text{cm}^2/\text{Vs nm}$

To activate only the thermionic emission term of the KVM ([Equation 389](#) and [Equation 391](#)) with available options, you can specify the following in the `Physics` section of the command file:

```
Mobility ( BalMob(KVM Fermi Frensley) )
```

where the `KVM` option must be used and activates the model, the `Fermi` option activates the Fermi–Dirac correction ([Equation 392](#)), and the `Frensley` option activates [Equation 393](#) for the final mobility  $\mu$  (if not specified, Matthiessen’s rule is used). To activate the total KVM with the free carrier acceleration term ([Equation 389](#), [Equation 390](#), and [Equation 391](#)) the `KVM(Full)` option must be used.

The KVM also has the same fitting parameter  $k_{300}$  that must be specified in the parameter file (see [Table 80](#)) and defines the effective mass  $m^*$  needed in [Equation 389](#) and [Equation 390](#). In addition, [Table 81](#) lists the KVM-specific parameters that are accessible in the `BalMob` parameter set in the parameter file.

Table 81      *BalMob model: KVM default coefficients for silicon*

Symbol	Parameter name	Electrons	Holes	Unit
$\alpha_T$	Tfactor	2	2	1
$\alpha_B$	Bfactor	0.3	0.3	1
$\alpha_0$	v0factor	1	1	1

---

## Using the PMI Fitting Parameter in the Ballistic Mobility Model

A PMI model for the fitting parameter  $k_{300}$  can be used in the ballistic mobility model. To activate this PMI model, specify the following in the `Physics` section of the command file:

```
Mobility ( BalMob(KVM pmi_parameterK_model_name) )
```

For an example of this PMI model, see [Fitting Parameter for Ballistic Mobility on page 1284](#). You can also use any variables computed by a mobility stress factor of the PMI model (see

## Chapter 15: Mobility

General Model for Mobility Degradation by Traps in the Bulk and at Semiconductor Interfaces

[Mobility Stress Factor on page 1377](#)) as dependent parameters to recalculate  $k_{300}$ . You can specify the following in the `Physics` section of the command file:

```
Mobility (
    BalMob(KVM pmi_parameterK_model_name)
)
Piezo (
    Model(
        Mobility(
            pmi_parameterK_model
            pmi_effectiveStress_model
        )
    )
)
```

---

## General Model for Mobility Degradation by Traps in the Bulk and at Semiconductor Interfaces

Traps at interfaces, grain boundaries, and in the bulk of a semiconductor cause mobility degradation by Coulomb scattering. The bulk trap Coulomb scattering (BTCS) model can model trap-related Coulomb scattering. It includes the dependency on the trap concentration, the influence of the screening by mobile charge carriers, and the weakening of Coulomb scattering with increasing temperature. [Equation 394](#) relates the trap concentration to the mobility degradation component, which is then combined using Matthiessen's rule to the total mobility:

$$\mu_C = \frac{\mu_1 \frac{T}{300 \text{ K}}^k \left[ \frac{c}{c_{\text{trans}} \frac{N_{A,D} + N_1}{10^{18} \text{ cm}^{-3}}} \right]^{\gamma_1} \left[ \frac{N_C}{N_0} \right]^{\eta_1}}{\left[ \frac{N_{A,D} + N_2}{10^{18} \text{ cm}^{-3}} \right]^{\gamma_2} \left[ \frac{N_C}{N_0} \right]^{\eta_2} \exp \left[ -\frac{r}{l_{\text{crit}}} \right]} \quad (394)$$

where:

- $N_C$  represents the total trapped charge.
- $c$  denotes the electron concentration  $n$  or the hole concentration  $p$ .
- $N_{A,D}$  is the acceptor or donor doping concentration for electron or hole mobility, respectively.
- $T$  is the device temperature.

## Chapter 15: Mobility

### Incomplete Ionization–Dependent Mobility Models

- $l_{\text{crit}}$  is a characteristic length that controls the influence of traps in the neighborhood  $r$  of a certain mesh point, on the mobility degradation. It is relevant only for the nonlocal option of the model.
- $\mu_1, k, c_{\text{trans}}, \gamma_1, \gamma_2, v, \eta_1, \eta_2, N_0, N_1$ , and  $N_2$  are fit parameters.

The model has the following options:

- The local option (default) calculates the mobility degradation at one mesh point caused by the trap concentration at that mesh point.
- The nonlocal option takes the mobility lowering caused by the traps located at a certain distance from the mesh point at which the mobility degradation is calculated into account. This is done by using Matthiessen's rule. The parameter `NonlocalLengthLimit` defines the corresponding radius of influence (in cm). Traps that are beyond this distance do not contribute to the mobility degradation at a given mesh point.

For example, to use the model for the region `R.channel`, specify:

```
Physics (Region="R.Channel") {
    Mobility( ... BTCSMob(Nonlocal NonlocalLengthLimit=1e-4) )
}
```

This statement switches on the nonlocal option of the model using 1.0e-4 cm for the `NonlocalLengthLimit` parameter.

The following parameters are available in the parameter file:

```
BTCSMobility {
    mul = 40, 40                      #[cm^2/Vs]
    T_exp = 1, 1                        #[1]
    c_trans = 1.0000e+18, 1.0000e+18   #[cm^(-3)]
    c_exp = 1.5, 1.5                   #[1]
    Nc_exp1 = 1, 1                      #[1]
    Nc_exp2 = 0.5, 0.5                 #[1]
    N0 = 1.0000e+18, 1.0000e+18       #[cm^(-3)]
    N1 = 1, 1                          #[cm^(-3)]
    N2 = 1, 1                          #[cm^(-3)]
    N_exp1 = 0.0000e+00, 0.0000e+00   #[1]
    N_exp2 = 0.0000e+00, 0.0000e+00   #[1]
    l_crit = 1.0000e-06, 1.0000e-06   #[cm]
```

---

## Incomplete Ionization–Dependent Mobility Models

Sentaurus Device supports incomplete ionization–dependent mobility models. To activate this dependence, specify the keyword `IncompleteIonization` in `Mobility` sections. The

## Chapter 15: Mobility

### Mobility Averaging

incomplete ionization model (see [Chapter 13 on page 339](#)) must also be activated. The `Physics` section for this case can be as follows:

```
Physics {
    IncompleteIonization
    Mobility ( Enormal IncompleteIonization )
}
```

In this case, for all equations that contain  $N_{A,0}$ ,  $N_{D,0}$ ,  $N_{\text{tot}}$ , Sentaurus Device uses  $N_A$ ,  $N_D$ ,  $N_i$ .

The following mobility models depend on incomplete ionization:

- Masetti model, see [Equation 253](#)
- Arora model, see [Equation 254](#)
- University of Bologna bulk model, see [Equation 258–Equation 260](#)
- Philips unified model, see [Equation 279](#) and [Equation 280](#)
- Lombardi model, see [Equation 285](#) and [Equation 288](#)
- IALMob model, see [Equation 290](#), [Equation 302](#), [Equation 303](#), [Equation 308](#), [Equation 310](#), and [Equation 311](#)
- University of Bologna inversion layer model, see [Equation 315–Equation 318](#)
- Coulomb degradation components, see [Equation 320](#)
- RCS model, see [Equation 325](#) and [Equation 326](#)

---

## Mobility Averaging

Sentaurus Device computes separate values of the mobility for each vertex of each semiconductor element of the mesh. To guarantee current conservation, these mobilities are then averaged to obtain either one value for each semiconductor element of the mesh or one value for each edge of each semiconductor element.

Element averaging is used by default and requires less memory than element–edge averaging. Element–edge averaging results in a smaller discretization error than element averaging, in particular, when the enhanced Lombardi model (see [Enhanced Lombardi Model on page 404](#)) with  $\alpha_{\perp} \neq 0$  is used. The time to compute the mobility is nearly identical for both approaches.

To select the mobility averaging approach, specify `eMobilityAveraging` and `hMobilityAveraging` in the `Math` section. The value is `Element` for element averaging, and `ElementEdge` for element–edge averaging.

## Chapter 15: Mobility

### Mobility Doping File

---

## Mobility Doping File

The `Grid` parameter in the `File` section of a command file can be used to read a TDR file that contains the device geometry, mesh, and doping.

By default, the doping read from this file is used in the mobility calculations previously described.

As an alternative, Sentaurus Device allows the donor and acceptor concentrations *for mobility calculations only* to be read from a separate TDR file. This is accomplished using the `MobilityDoping` parameter:

```
File {  
    Grid      = "mosfet.tdr"  
    MobilityDoping = "mosfet_mobility.tdr"  
}
```

Notes:

- The geometry and mesh in the `MobilityDoping` file must match the `Grid` file.
- If a `MobilityDoping` file is specified, then it deactivates the mobility dependency on `IncompleteIonization` if this mobility option is specified.
- Donor and acceptor concentrations read from a `MobilityDoping` file can be specified in the `Plot` section of the command file with `MobilityDonorConcentration` and `MobilityAcceptorConcentration`, respectively.
- For PMI models, the `MobilityDoping` file concentrations can be read using:

```
double Nd = ReadDoping("MobilityDonorConcentration")  
double Na = ReadDoping("MobilityAcceptorConcentration")
```

---

## Effective Mobility

It is often useful to know the effective channel mobility seen by carriers as well as other effective or averaged quantities in the channel. To simplify the task of obtaining such quantities, Sentaurus Device provides an `EffectiveMobility` current plot PMI that extracts several quantities of interest for both electrons and holes.

[Table 82](#) lists the electron quantities extracted by the `EffectiveMobility` PMI. The extracted hole quantities are analogous.

## Chapter 15: Mobility

### Effective Mobility

**Table 82** Electron quantities extracted with the EffectiveMobility PMI; hole quantities are analogous ( $\tilde{n} = n$  or  $n_{\text{SHE}}$  if UseSHEDensity = 0 or 1, respectively)

Name	Symbol	Start Point or Start Box method	Volume Box method	Unit
eSheetDensity	$n_{\text{sheet}}$	$\frac{s_{\max}}{\int_0^{s_{\max}} n(s) ds}$	$\frac{n(\mathbf{r}) d\Omega}{dS}$	$\text{cm}^{-2}$
eAverageDensity	$n_{\text{avg}}$	$\frac{1}{s_{\max}} \int_0^{s_{\max}} n(s) ds$	$\frac{n(\mathbf{r}) d\Omega}{d\Omega}$	$\text{cm}^{-3}$
eSHESheet Density	$n_{\text{SHE,sheet}}$	$\frac{s_{\max}}{\int_0^{s_{\max}} n_{\text{SHE}}(s) ds}$	$\frac{n_{\text{SHE}}(\mathbf{r}) d\Omega}{dS}$	$\text{cm}^{-2}$
eSHEAverage Density	$n_{\text{SHE,avg}}$	$\frac{1}{s_{\max}} \int_0^{s_{\max}} n_{\text{SHE}}(s) ds$	$\frac{n_{\text{SHE}}(\mathbf{r}) d\Omega}{d\Omega}$	$\text{cm}^{-3}$
eAverageField	$E_{\text{avg,n}}$	$\frac{1}{n_{\text{sheet}}} \int_0^{s_{\max}} E_{\perp}(s) \tilde{n}(s) ds$	$\frac{ E(\mathbf{r})  \tilde{n}(\mathbf{r}) d\Omega}{\tilde{n}(\mathbf{r}) d\Omega}$	$\text{V/cm}$
eEffectiveField	$E_{\text{eff,n}}$	$E_{\text{avg,n}} + \eta_e - \frac{1}{2} \frac{q}{\epsilon_s} \tilde{n}_{\text{sheet}}$	$E_{\text{avg,n}} + \eta_e - \frac{1}{2} \frac{q}{\epsilon_s} \tilde{n}_{\text{sheet}}$	$\text{V/cm}$
eChargeEffField	$E_{\text{eff,ch,n}}$	$\frac{q}{\epsilon_s} \int_0^{s_{\max}} (N_A - N_D - p) ds + \eta_e \tilde{n}_{\text{sheet}}$	$\frac{q}{\epsilon_s} \frac{(N_A - N_D - p) d\Omega}{dS} + \eta_e \tilde{n}_{\text{sheet}}$	$\text{V/cm}$
eMobility	$\mu_{\text{eff,n}}$	$\frac{1}{n_{\text{sheet}}} \int_0^{s_{\max}} \mu_n(s) \tilde{n}(s) ds$	$\frac{\mu_n(\mathbf{r}) \tilde{n}(\mathbf{r}) d\Omega}{\tilde{n}(\mathbf{r}) d\Omega}$	$\text{cm}^2/\text{Vs}$
eStressFactorXX	$\frac{\mu_{n,xx}}{\mu_{n0}}_{\text{eff}}$	$\frac{1}{n_{\text{sheet}} \mu_{\text{eff,n}}} \int_0^{s_{\max}} \frac{\mu_{n,xx}}{\mu_{n0}} \mu_n(s) \tilde{n}(s) ds$	$\frac{(\mu_{n,xx}/\mu_{n0}) \mu_n(\mathbf{r}) \tilde{n}(\mathbf{r}) d\Omega}{\mu_n(\mathbf{r}) \tilde{n}(\mathbf{r}) d\Omega}$	1

## Chapter 15: Mobility

### Effective Mobility

**Table 82** Electron quantities extracted with the EffectiveMobility PMI; hole quantities are analogous ( $n = n$  or  $n_{\text{SHE}}$  if UseSHEDensity = 0 or 1, respectively) (Continued)

Name	Symbol	Start Point or Start Box method	Volume Box method	Unit
eStressFactorYY	$\frac{\mu_{n,yy}}{\mu_{n0}}_{\text{eff}}$	$\frac{1}{n_{\text{sheet}}\mu_{\text{eff},n}} \int_0^{s_{\max}} \frac{\mu_{n,yy}}{\mu_{n0}} \mu_n(s) \tilde{n}(s) ds$	$\frac{(\mu_{n,yy}/\mu_{n0})\mu_n(r)\tilde{n}(r)d\Omega}{\mu_n(r)\tilde{n}(r)d\Omega}$	1
eStressFactorZZ	$\frac{\mu_{n,zz}}{\mu_{n0}}_{\text{eff}}$	$\frac{1}{n_{\text{sheet}}\mu_{\text{eff},n}} \int_0^{s_{\max}} \frac{\mu_{n,zz}}{\mu_{n0}} \mu_n(s) \tilde{n}(s) ds$	$\frac{(\mu_{n,zz}/\mu_{n0})\mu_n(r)\tilde{n}(r)d\Omega}{\mu_n(r)\tilde{n}(r)d\Omega}$	1

## EffectiveMobility PMI Methods

The `EffectiveMobility` PMI provides different methods for extracting quantities from the device:

- The *Start Point method* extracts quantities at a single point in the channel. The method is invoked by specifying the `Start` parameter (see [Using the EffectiveMobility PMI](#)) to identify the starting location for a line that will pass through the structure. The extracted quantities will be obtained from integrals along this line. By default, the selected `Start` location will snap to the nearest semiconductor–insulator interface vertex, and the line will be normal to the interface.
- The *Start Box method* is an extension of the Start Point method and is used to extract quantities over an extended region of the channel. The method is invoked by specifying the `StartBoxMin` and `StartBoxMax` parameters to surround the portion of the semiconductor–insulator interface where the extraction will occur. All interface vertices enclosed in the start box will be used as starting locations for line integrals through the device. The reported extracted quantities will be an interface-area weighted average of all the calculated line integrals.
- In contrast to the previous method, the quantities extracted with the *Volume Box method* will be obtained by performing volume integrals over a selected portion of the device. The method is invoked by specifying the `BoxMin` and `BoxMax` parameters to identify where the integration should occur. The specified box should surround both the interface and volume of interest. The size of the semiconductor–insulator interface area enclosed in the box will be extracted and used in the calculations whenever a sheet density (#/cm<sup>2</sup>) is required.

---

## Using the EffectiveMobility PMI

To include the quantities listed in [Table 82 on page 466](#) in the current plot file, specify one or more instances of the `EffectiveMobility` PMI in the `CurrentPlot` section of the command file. The syntax and options for the `EffectiveMobility` PMI are:

```
CurrentPlot {
    PMIModel (
        Name = "EffectiveMobility"
        [GroupName=<string>
         [Start= (x0, y0, z0) [SnapToNode= 0 | 1]] |
         [StartBoxMin= (xmin, ymin, zmin) StartBoxMax= (xmax, ymax, zmax) ]
         [Direction = (dx, dy, dz)] [Depth= smax] ] |
         [BoxMin= (xmin, ymin, zmin) BoxMax= (xmax, ymax, zmax)
          [Width=<float>]]
         [Region= (<string1>, <string2>, ...)]
         [ChannelType= -1 | 1 | 0]
         [eta_e= ηe] [eta_h= ηh] [epsilon= εs/ε0]
         [UseSHEDensity= 0 | 1]
     )
}
```

The parameters are described here:

- `GroupName` is the name of the data group for the extracted quantities. A data group with this name will appear in Sentaurus Visual or Inspect when a current plot file (`*.plt` file) is loaded. The default group name is "Channel". If there is more than one `EffectiveMobility` PMI used in a `CurrentPlot` section, `GroupName` must be used to distinguish between different sets of results.
- `Start` specifies the starting location used in the Start Point method. All coordinates must be specified in  $\mu\text{m}$ . Only the coordinates corresponding to the dimensionality of the structure are required (for example, only  $x_0$  and  $y_0$  in two dimensions).
- `SnapToNode` is used to snap the `Start` location to the nearest semiconductor–insulator interface vertex. Specify `SnapToNode=1` to snap (default). Specify `SnapToNode=0` to not snap.
- `Direction` is used to specify a direction vector for the line along which the line integrals will be performed. By default, the direction for lines initiated at a semiconductor–insulator interface vertex will be taken as normal to the interface. If `Direction` is specified, the units are arbitrary, as the specified direction vector will be normalized.
- `Depth` is the line integration distance in  $\mu\text{m}$ . The default is 0.5  $\mu\text{m}$ . The actual integration distance can be shorter if the line encounters a structure boundary, a non-semiconductor material, or a region not included in the `Region` list (see below). `Depth` is usually chosen so that contributions to the integral beyond a distance of `Depth` are negligible.

## Chapter 15: Mobility

### References

- `StartBoxMin` and `StartBoxMax` specify the minimum and maximum coordinates for a box that surrounds the semiconductor–insulator interface starting points used in the Start Box method. All coordinates are specified in  $\mu\text{m}$ .
- `BoxMin` and `BoxMax` specify the minimum and maximum coordinates for a box that surrounds the semiconductor volume and interface used in the Volume Box method. All coordinates are specified in  $\mu\text{m}$ .
- `Width` represents an interface length in two dimensions or an interface area in three dimensions, and is used when sheet densities are required for the extracted quantities. `Width` must be specified in  $\mu\text{m}$  for 2D structures and  $\mu\text{m}^2$  for 3D structures. If `Width` is not specified, the interface length or area is extracted automatically from the portion of the interface enclosed within the user-specified box. A default value of `Width = 0.04  $\mu\text{m}$`  in two dimensions or `0.0016  $\mu\text{m}^2$`  in three dimensions will be used if an interface length or area cannot be found.
- `Region` can optionally be used to specify a list of region names where the extraction calculations will be confined. If `Region` is not specified, all semiconductor regions will be considered.
- `ChannelType` specifies which quantities are extracted as follows:
  - Specify `ChannelType=-1` to extract electron quantities.
  - Specify `ChannelType=1` to extract hole quantities.
  - Specify `ChannelType=0` to extract both quantities.
- `eta_e`, `eta_h`, and `epsilon` are parameters used in the calculation of effective fields. Their default values are `eta_e=0.5`, `eta_h=0.333333`, and `epsilon=11.7`.
- `UseSHEDensity` specifies whether the standard carrier densities (`UseSHEDensity=0`) or the SHE carrier densities (`UseSHEDensity=1`) are used as the weighting functions in the integral calculations. The default is `UseSHEDensity=0`.

---

## References

- [1] C. Lombardi *et al.*, “A Physically Based Mobility Model for Numerical Simulation of Nonplanar Devices,” *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 11, pp. 1164–1171, 1988.
- [2] G. Masetti, M. Severi, and S. Solmi, “Modeling of Carrier Mobility Against Carrier Concentration in Arsenic-, Phosphorus-, and Boron-Doped Silicon,” *IEEE Transactions on Electron Devices*, vol. ED-30, no. 7, pp. 764–769, 1983.
- [3] N. D. Arora, J. R. Hauser, and D. J. Roulston, “Electron and Hole Mobilities in Silicon as a Function of Concentration and Temperature,” *IEEE Transactions on Electron Devices*, vol. ED-29, no. 2, pp. 292–295, 1982.

## Chapter 15: Mobility

### References

- [4] S. Reggiani *et al.*, “A Unified Analytical Model for Bulk and Surface Mobility in Si n- and p-Channel MOSFET’s,” in *Proceedings of the 29th European Solid-State Device Research Conference (ESSDERC)*, Leuven, Belgium, pp. 240–243, September 1999.
- [5] S. Reggiani *et al.*, “Electron and Hole Mobility in Silicon at Large Operating Temperatures—Part I: Bulk Mobility,” *IEEE Transactions on Electron Devices*, vol. 49, no. 3, pp. 490–499, 2002.
- [6] M. S. Shur, “Low Ballistic Mobility in Submicron HEMTs,” *IEEE Electron Device Letters*, vol. 23, no. 9, pp. 511–513, 2002.
- [7] A. Erlebach, K. H. Lee, and F. M. Bufler, “Empirical Ballistic Mobility Model for Drift-Diffusion Simulation,” in *Proceedings of the 46th European Solid-State Device Research Conference (ESSDERC)*, Lausanne, Switzerland, pp. 420–423, September 2016.
- [8] S. C. Choo, “Theory of a Forward-Biased Diffused-Junction P-L-N Rectifier—Part I: Exact Numerical Solutions,” *IEEE Transactions on Electron Devices*, vol. ED-19, no. 8, pp. 954–966, 1972.
- [9] N. H. Fletcher, “The High Current Limit for Semiconductor Junction Devices,” *Proceedings of the IRE*, vol. 45, no. 6, pp. 862–872, 1957.
- [10] A. Schenk, *Advanced Physical Models for Silicon Device Simulation*, Wien: Springer, 1998.
- [11] D. B. M. Klaassen, “A Unified Mobility Model for Device Simulation—I. Model Equations and Concentration Dependence,” *Solid-State Electronics*, vol. 35, no. 7, pp. 953–959, 1992.
- [12] M. N. Darwish *et al.*, “An Improved Electron and Hole Mobility Model for General Purpose Device Simulation,” *IEEE Transactions on Electron Devices*, vol. 44, no. 9, pp. 1529–1538, 1997.
- [13] S. A. Mujtaba, *Advanced Mobility Models for Design and Simulation of Deep Submicrometer MOSFETs*, PhD thesis, Stanford University, Stanford, CA, USA, December 1995.
- [14] W. Hänsch and M. Miura-Mattausch, “The hot-electron problem in small semiconductor devices,” *Journal of Applied Physics*, vol. 60, no. 2, pp. 650–656, 1986.
- [15] S. Takagi *et al.*, “On the Universality of Inversion Layer Mobility in Si MOSFET’s: Part I—Effects of Substrate Impurity Concentration,” *IEEE Transactions on Electron Devices*, vol. 41, no. 12, pp. 2357–2362, 1994.
- [16] G. Baccarani, *A Unified mobility model for Numerical Simulation, Parasitics Report*, DEIS-University of Bologna, Bologna, Italy, 1999.
- [17] H. Tanimoto *et al.*, “Modeling of Electron Mobility Degradation for HfSiON MISFETs,” in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, Monterey, CA, USA, September 2006.

## Chapter 15: Mobility

### References

- [18] W. J. Zhu and T. P. Ma, "Temperature Dependence of Channel Mobility in HfO<sub>2</sub>-Gated NMOSFETs," *IEEE Electron Device Letters*, vol. 25, no. 2, pp. 89–91, 2004.
- [19] S. Reggiani *et al.*, "Low-Field Electron Mobility Model for Ultrathin-Body SOI and Double-Gate MOSFETs With Extremely Small Silicon Thicknesses," *IEEE Transactions on Electron Devices*, vol. 54, no. 9, pp. 2204–2212, 2007.
- [20] Y. Zhao, A. Liao, and E. Pop, "Multiband Mobility in Semiconducting Carbon Nanotubes," *IEEE Electron Device Letters*, vol. 30, no. 10, pp. 1078–1080, 2009.
- [21] C. Jacoboni *et al.* "A Review of Some Charge Transport Properties of Silicon," *Solid-State Electronics*, vol. 20, no. 2, pp. 77–89, 1977.
- [22] Y. Kawaguchi and S. Kawaji, "Lattice Scattering Mobility of n-Inversion Layers in Si(100) at Low Temperatures," *Surface Science*, vol. 98, no. 1–3, pp. 211–217, 1980.
- [23] C. Canali *et al.*, "Electron and Hole Drift Velocity Measurements in Silicon and Their Empirical Relation to Electric Field and Temperature," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 11, pp. 1045–1047, 1975.
- [24] D. M. Caughey and R. E. Thomas, "Carrier Mobilities in Silicon Empirically Related to Doping and Field," *Proceedings of the IEEE*, vol. 55, no. 12, pp. 2192–2193, 1967.
- [25] J. J. Barnes, R. J. Lomax, and G. I. Haddad, "Finite-Element Simulation of GaAs MESFET's with Lateral Doping Profiles and Submicron Gates," *IEEE Transactions on Electron Devices*, vol. ED-23, no. 9, pp. 1042–1048, 1976.
- [26] M. Farahmand *et al.*, "Monte Carlo Simulation of Electron Transport in the III-Nitride Wurtzite Phase Materials System: Binaries and Ternaries," *IEEE Transactions on Electron Devices*, vol. 48, no. 3, pp. 535–542, 2001.
- [27] V. M. Polyakov and F. Schwierz, "Influence of Electron Mobility Modeling on DC *I*–*V* Characteristics of WZ-GaN MESFET," *IEEE Transactions on Electron Devices*, vol. 48, no. 3, pp. 512–516, 2001.
- [28] B. Meinerzhagen and W. L. Engl, "The Influence of the Thermal Equilibrium Approximation on the Accuracy of Classical Two-Dimensional Numerical Modeling of Silicon Submicrometer MOS Transistors," *IEEE Transactions on Electron Devices*, vol. 35, no. 5, pp. 689–697, 1988.
- [29] H. Cordes *et al.*, "One-dimensional hopping transport in disordered organic solids. I. Analytic calculations," *Physical Review B*, vol. 63, p. 094201, 2001.
- [30] O. Penzin *et al.*, "Kinetic Velocity Model to Account for Ballistic Effects in the Drift-Diffusion Transport Approach," *IEEE Transactions on Electron Devices*, vol. 64, no. 11, pp. 4599–4606, 2017.
- [31] W. R. Frensley, "Barrier-Limited Transport in Semiconductor Devices," *IEEE Transactions on Electron Devices*, vol. ED-30, no. 12, pp. 1619–1623, 1983.
- [32] F. M. Bufler and W. Fichtner, "Hole and electron transport in strained Si: Orthorhombic versus biaxial tensile strain," *Applied Physics Letters*, vol. 81, no. 1, pp. 82–84, 2002.

## Chapter 15: Mobility

### References

- [33] M. T. Currie *et al.*, “Carrier mobilities and process stability of strained Si n- and p-MOSFETs on SiGe virtual substrates,” *Journal of Vacuum Science & Technology B*, vol. 19, no. 6, pp. 2268–2279, 2001.
- [34] C. W. Leitz *et al.*, “Hole mobility enhancements and alloy scattering-limited mobility in tensile strained Si/SiGe surface channel metal–oxide–semiconductor field-effect transistors,” *Journal of Applied Physics*, vol. 92, no. 7, pp. 3745–3751, 2002.
- [35] F. M. Bufler, *Full-Band Monte Carlo Simulation of Electrons and Holes in Strained Si and SiGe*, München: Herbert Utz Verlag, 1998.
- [36] R. Braunstein, A. R. Moore, and F. Herman, “Intrinsic Optical Absorption in Germanium-Silicon Alloys,” *Physical Review*, vol. 109, no. 3, pp. 695–710, 1958.

# 16

## Generation–Recombination

---

*This chapter describes the generation–recombination processes that can be modeled in Sentaurus Device.*

Generation–recombination processes exchange carriers between the conduction band and the valence band. They are very important in device physics, in particular, for bipolar devices. This chapter describes the generation–recombination models available in Sentaurus Device. Most models are local in the sense that their implementation (sometimes in contrast to reality) does not involve spatial transport of charge. For each individual generation or recombination process, the electrons and holes involved appear or vanish at the same location. The only exceptions are one trap-assisted tunneling model and one band-to-band tunneling model (see [Dynamic Nonlocal Path Trap-Assisted Tunneling on page 481](#) and [Dynamic Nonlocal Path Band-to-Band Tunneling Model on page 529](#)). For models that couple the conduction and valence bands and account for spatial transport of charge, see [Nonlocal Tunneling for Traps on page 576](#) and [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 826](#).

---

### Shockley–Read–Hall Recombination

Recombination through deep defect levels in the gap is usually called Shockley–Read–Hall (SRH) recombination. In Sentaurus Device, the following form is implemented:

$$R_{\text{net}}^{\text{SRH}} = \frac{np - n_{\text{i,eff}}^2}{\tau_p(n + n_1) + \tau_n(p + p_1)} \quad (395)$$

with:

$$n_1 = n_{\text{i,eff}} \exp \left[ \frac{E_{\text{trap}}}{kT} \right] \quad (396)$$

$$p_1 = n_{\text{i,eff}} \exp \left[ \frac{-E_{\text{trap}}}{kT} \right] \quad (397)$$

where  $E_{\text{trap}}$  is the difference between the defect level and intrinsic level. The variable  $E_{\text{trap}}$  is accessible in the parameter file. The silicon default value is  $E_{\text{trap}} = 0$ .

## Chapter 16: Generation–Recombination

### Shockley–Read–Hall Recombination

The lifetimes  $\tau_n$  and  $\tau_p$  are modeled as a product of a doping-dependent (see [SRH Doping Dependence on page 474](#)), field-dependent (see [SRH Field Enhancement on page 477](#)), and temperature-dependent (see [SRH Temperature Dependence on page 476](#)) factor:

$$\tau_c = \tau_{\text{dop}} \frac{f(T)}{1 + g_c(F)} \quad (398)$$

where  $c = n$  or  $c = p$ . For an additional density dependency of the lifetimes, see [Trap-Assisted Auger Recombination Model on page 485](#).

For simulations that use Fermi statistics (see [Fermi Statistics on page 233](#)) or quantization (see [Chapter 14 on page 348](#)), Equation 395 must be generalized. The modified equation reads:

$$R_{\text{net}}^{\text{SRH}} = \frac{np - \gamma_n \gamma_p n_{\text{eff}}^2}{\tau_p(n + \gamma_n n_1) + \tau_n(p + \gamma_p p_1)} \quad (399)$$

where  $\gamma_n$  and  $\gamma_p$  are given by [Equation 47](#) and [Equation 48](#).

---

## Using SRH Recombination

Generation–recombination models are selected in the `Physics` section as an argument to the `Recombination` keyword:

```
Physics{ Recombination( <arguments> ) ... }
```

The SRH model is activated by specifying the `SRH` argument:

```
Physics{ Recombination( SRH ... ) ... }
```

The keyword for plotting the SRH recombination rate is:

```
Plot{ ...
      SRHRecombination
}
```

---

## SRH Doping Dependence

Doping dependence of SRH lifetimes is modeled in Sentaurus Device with the Scharfetter relation:

$$\tau_{\text{dop}}(N_{A,0} + N_{D,0}) = \tau_{\text{min}} + \frac{\tau_{\text{max}} - \tau_{\text{min}}}{1 + \frac{N_{A,0} + N_{D,0}}{N_{\text{ref}}}} \gamma \quad (400)$$

Such a dependence arises from experimental data [1] and the theoretical conclusion that the solubility of a fundamental acceptor-type defect (probably a divacancy (E5) or a vacancy complex) is strongly correlated to the doping density [2][3][4]. [Table 83 on page 477](#) lists the

## Chapter 16: Generation–Recombination

Shockley–Read–Hall Recombination

default values. The Scharfetter relation is used when the option `DopingDependence` is specified for SRH recombination:

```
Physics{ Recombination( SRH( DopingDependence ... ) ... ) ... }
```

Without the option `DopingDependence`,  $\tau = \tau_{\max}$  is used.

---

## Lifetime Profiles From Files

Sentaurus Device can use spatial lifetime profiles provided by a file. Such profiles can be precomputed or generated manually by an editor such as Sentaurus Structure Editor (see the *Sentaurus™ Structure Editor User Guide*).

The names of the datasets for electron and hole lifetimes must be `eLifetime` and `hLifetime`, respectively. They are loaded from a file named by the keyword `LifeTime` in the `File` section:

```
File {
    Grid      = "MyDev_msh.tdr"
    LifeTime = "MyDev_msh.tdr"
    ...
}
```

For each grid point, the values defined by the lifetime profile are used as  $\tau_{\max}$  in [Equation 400](#).

Lifetime data can be in a separate file from the doping, but it must correspond to the same grid.

---

## Improved Nakagawa Model

The Scharfetter relation can be expanded to be:

$$\tau_{\text{dop}}(N_{A,0} + N_{D,0}) = \frac{\tau_{\max} + \tau_{\min} \frac{N_{A,0} + N_{D,0}}{N_{\text{ref}}}^{\gamma}}{1 + \frac{\tau_{\max} N_{A,0} + N_{D,0}}{\tau_0 N_{\text{ref}}}^{\gamma}} \quad (401)$$

Here,  $\tau_0$  is an additional fitting parameter. By default,  $\tau_{\min} = 0$ .

Then, [Equation 401](#) simplifies to:

$$\tau_{\text{dop}}(N_{A,0} + N_{D,0}) = \frac{\tau_{\max}}{1 + \frac{\tau_{\max} N_{A,0} + N_{D,0}}{\tau_0 N_{\text{ref}}}^{\gamma}} \quad (402)$$

## Chapter 16: Generation–Recombination

Shockley–Read–Hall Recombination

With another default,  $\gamma = 1$ , [Equation 402](#) is further simplified to:

$$\frac{1}{\tau_{\text{dop}}(N_{A,0} + N_{D,0})} = \frac{1}{\tau_{\max}} + \frac{1}{\tau_0} \cdot \frac{N_{A,0} + N_{D,0}}{N_{\text{ref}}} \quad (403)$$

This is the original equation in [5]. With  $\tau_0 = \tau_{\max}$ , it corresponds to [Equation 400](#) with  $\tau_{\min} = 0$ ,  $\gamma = 1$ . [Table 83 on page 477](#) lists the default values.

To activate the Nakagawa model, an additional option `Nakagawa` is specified in the `SRH(DopingDependence)` statement:

```
Physics {
    Recombination( SRH( DopingDependence ( Nakagawa )... ) ... )
    ...
}
```

## SRH Temperature Dependence

There is no consensus on the temperature dependence of SRH lifetimes. This appears to originate from a different understanding of lifetime. From measurements of the recombination lifetime [6][7]:

$$\tau = \frac{\delta n}{R} \quad (404)$$

in power devices ( $\delta n$  is the excess carrier density under neutral conditions,  $\delta n = \delta p$ ), it was concluded that the lifetime increases with rising temperature.

Such a dependence was modeled either by a power law [6][7]:

$$\tau(T) = \tau_0 \left[ \frac{T}{300 \text{ K}} \right]^{\alpha} \quad (405)$$

or an exponential expression of the form:

$$\tau(T) = \tau_0 e^{C \frac{T}{300 \text{ K}} - 1} \quad (406)$$

A calculation using the low-temperature approximation of multiphonon theory [8] gives:

$$(\tau_{\text{SRH}}(T) = \tau_{\text{SRH}}(300 \text{ K}) \cdot f(T)) \text{ with } fT = \left[ \frac{T}{300 \text{ K}} \right]^{T_\alpha} \quad (407)$$

with  $T_\alpha = -3/2$ , which is the expected decrease of minority carrier lifetimes with rising temperature. Since the temperature behavior strongly depends on the nature of the recombination centers, there is no universal law  $\tau_{\text{SRH}}(T)$ .

## Chapter 16: Generation–Recombination

### Shockley–Read–Hall Recombination

In Sentaurus Device, the power law model ([Equation 405](#)) is activated with the `TempDependence` keyword in the `SRH` statement:

```
Physics { Recombination( SRH( TempDependence ... ) ... ) ... }
```

Additionally, Sentaurus Device supports an exponential model for  $f(T)$ :

$$f(T) = e^{-\frac{C}{300 \text{ K}} - 1} \quad (408)$$

This model is activated with the keyword `ExpTempDependence`:

```
Physics { Recombination( SRH( ExpTempDependence ... ) ... ) ... }
```

---

## SRH Doping- and Temperature-Dependent Parameters

All the parameters of the doping- and temperature-dependent SRH recombination models are accessible in the `Scharfetter` parameter set.

*Table 83 Parameters for doping- and temperature-dependent SRH lifetimes*

Symbol	Parameter name	Default value		Unit
		Electrons	Holes	
$\tau_{\min}$	taumin	0	0	s
$\tau_{\max}$	taumax	$1 \times 10^{-5}$	$3 \times 10^{-6}$	s
$\tau_0$	tau0	$1 \times 10^{-5}$	$3 \times 10^{-6}$	s
$N_{\text{ref}}$	Nref	$1 \times 10^{16}$	$1 \times 10^{16}$	$\text{cm}^{-3}$
$\gamma$	gamma	1	1	1
$T_\alpha$	Talpha	-1.5	-1.5	1
$C$	Tcoeff	2.55	2.55	1
$E_{\text{trap}}$	Etrap	0	0	eV

---

## SRH Field Enhancement

Field enhancement reduces SRH recombination lifetimes in regions of strong electric fields. It must not be neglected if the electric field exceeds a value of approximately  $3 \times 10^5$  V/cm in certain regions of the device. For example, the I–V characteristics of reverse-biased p-n

## Chapter 16: Generation–Recombination

### Shockley–Read–Hall Recombination

junctions are extremely sensitive to defect-assisted tunneling, which causes electron–hole pair generation before band-to-band tunneling or avalanche generation sets in. Therefore, it is recommended that field enhancement is included in the simulation of drain reverse leakage and substrate currents in MOS transistors.

Sentaurus Device provides the following field enhancement models:

- [Schenk Trap-Assisted Tunneling Model on page 479](#)
- [Hurkx Trap-Assisted Tunneling Model on page 480](#)

The Hurkx model is also available for trap capture and emission rates (see [Hurkx Model for Cross Sections on page 557](#)).

## Using Field Enhancement

Parameters in the `ElectricField` option of the `SRH` statement activate the local field-dependence of SRH lifetimes:

```
SRH( ...
    ElectricField (
        Lifetime = Constant | Hurkx | Schenk
        DensityCorrection = Local | None
    )
)
```

The `Lifetime` parameter selects the lifetime model:

- The default is `Constant`, which is for field-independent lifetimes.
- `Hurkx` selects the Hurkx lifetimes.
- `Schenk` selects the Schenk lifetimes.

The `DensityCorrection` parameter defaults to `None`. A value of `Local` selects the model described in [Density Correction for Schenk and Hurkx Trap-Assisted Tunneling Models on page 481](#).

For backward compatibility:

- `SRH (Tunneling)` selects `Lifetime=Schenk` and `DensityCorrection=Local`.
- `SRH(Tunneling(Hurkx))` selects `Lifetime=Hurkx` and `DensityCorrection=None`.

### Note:

The inclusion of the Schenk or Hurkx model might lead to convergence problems. In such cases, try to specify the `NoSRHperPotential` flag in the `Math` section, which causes Sentaurus Device to exclude derivatives of  $g(F)$  with respect to the potential from the Jacobian matrix.

## Schenk Trap-Assisted Tunneling Model

The field dependence of the recombination rate is taken into account by the field enhancement factors:

$$[1 + g(F)]^{-1} \quad (409)$$

of the SRH lifetimes [8] (see [Equation 398](#)).

In the case of electrons,  $g(F)$  has the form:

$$g_n(F) = 1 + \frac{(\hbar\Theta)^{3/2} \sqrt{E_t - E_0}}{E_0 \hbar\omega_0} \frac{-\frac{1}{2} \frac{(\hbar\Theta)^{3/4} (E_t - E_0)^{1/4}}{2 \sqrt{E_t E_0}} \frac{\hbar\Theta}{kT}^{\frac{3}{2}}}{\times \exp \left( -\frac{E_t - E_0}{\hbar\omega_0} + \frac{\hbar\omega_0 - kT}{2\hbar\omega_0} + \frac{2E_t + kT}{2\hbar\omega_0} \ln \frac{E_t}{\varepsilon_R} - \frac{E_0}{\hbar\omega_0} \ln \frac{E_0}{\varepsilon_R} + \frac{E_t - E_0}{kT} - \frac{4}{3} \frac{(E_t - E_0)^{\frac{3}{2}}}{\hbar\Theta} \right)} \quad (410)$$

where  $E_0$  denotes the energy of an optimum horizontal transition path, which depends on field strength and temperature in the following way:

$$E_0 = 2\sqrt{\varepsilon_F} [\sqrt{\varepsilon_F + E_t + \varepsilon_R} - \sqrt{\varepsilon_F}] - \varepsilon_R, \quad \varepsilon_F = \frac{(2\varepsilon_R kT)^2}{(\hbar\Theta)^3} \quad (411)$$

In this expression,  $\varepsilon_R = S\hbar\omega_0$  is the lattice relaxation energy,  $S$  is the Huang–Rhys factor,  $\hbar\omega_0$  is the effective phonon energy,  $E_t$  is the energy level of the recombination center (thermal depth), and  $\Theta = (q^2 F^2 / 2\hbar m_{\Theta,n})^{1/3}$  is the electro-optical frequency.

The mass  $m_{\Theta,n}$  is the electron tunneling mass in the field direction and is given in the parameter file. The expression for holes follows from [Equation 410](#) by replacing  $m_{\Theta,n}$  with  $m_{\Theta,p}$  and  $E_t$  with  $E_{g,\text{eff}} - E_t$ .

For electrons,  $E_t$  is related to the defect level  $E_{\text{trap}}$  of [Equation 396](#) and [Equation 397](#) by:

$$E_t = \frac{1}{2} E_{g,\text{eff}} + \frac{3}{4} kT \ln \frac{m_n}{m_p} + E_{\text{trap}} - (32R_C \hbar^3 \Theta^3)^{1/4} \quad (412)$$

where the effective Rydberg constant  $R_C$  is:

$$R_C = m_C \frac{Z^2}{\varepsilon^2} R_y \quad (413)$$

where  $R_y$  is the Rydberg energy (13.606 eV),  $\varepsilon$  is the relative dielectric constant, and  $Z$  is a fit parameter.

## Chapter 16: Generation–Recombination

Shockley–Read–Hall Recombination

For holes,  $E_t$  is given by:

$$E_t = \frac{1}{2}E_{g,eff} - \frac{3}{4}kT \ln \left[ \frac{m_t}{m_p} + E_{trap} \right] - (32R_V\hbar^3\Theta^3)^{1/4} \quad (414)$$

Note that  $E_{trap}$  is measured from the intrinsic level and not from the midgap. The zero-field lifetime  $\tau_{SRH}$  is defined by [Equation 400](#).

## Hurkx Trap-Assisted Tunneling Model

The following equations apply to electrons and holes. Lifetimes and capture cross-sections become functions of the trap-assisted tunneling factor  $g(F) = \Gamma_{tat}$ :

$$\tau = \tau_0 / (1 + \Gamma_{tat}) \quad , \quad \sigma = \sigma_0 (1 + \Gamma_{tat}) \quad (415)$$

where  $\Gamma_{tat}$  is given by:

$$\Gamma_{tat} = \int_0^{\tilde{E}_n} \exp \left[ u - \frac{2}{3} \frac{\sqrt{u^3}}{\tilde{E}} \right] du \quad (416)$$

with the approximate solutions:

$$\begin{aligned} & \sqrt{\pi} \tilde{E} \cdot \exp \left[ \frac{1}{3} \tilde{E}^2 \right] 2 - \operatorname{erfc} \left[ \frac{1}{2} \frac{\tilde{E}_n}{\tilde{E}} - \tilde{E} \right], \quad \tilde{E} \leq \sqrt{\tilde{E}_n} \\ \Gamma_{tat} \approx & \sqrt{\pi} \tilde{E} \cdot \tilde{E}_n^{1/4} \exp \left[ -\tilde{E}_n + \tilde{E} \sqrt{\tilde{E}_n} + \frac{1}{3} \frac{\sqrt{\tilde{E}_n^3}}{\tilde{E}} \right] \operatorname{erfc} \left[ \tilde{E}_n^{1/4} \sqrt{\tilde{E}} - \tilde{E}_n^{3/4} / \sqrt{\tilde{E}} \right], \quad \tilde{E} > \sqrt{\tilde{E}_n} \end{aligned} \quad (417)$$

where  $\tilde{E}$  and  $\tilde{E}_n$  are respectively defined as:

$$\tilde{E} = \frac{E}{E_0} \quad \text{where} \quad E_0 = \frac{\sqrt{8m_0 m_t k^3 T^3}}{q\hbar} \quad (418)$$

$$\begin{aligned} & 0, \quad kT \ln \frac{n}{n_i} > 0.5E_g \\ \tilde{E}_n = \frac{E_n}{kT} = & \frac{0.5E_g}{kT} - \ln \frac{n}{n_i}, \quad E_{trap} \leq kT \ln \frac{n}{n_i} \leq 0.5E_g \\ & \frac{0.5E_g}{kT} - \frac{E_{trap}}{kT}, \quad E_{trap} > kT \ln \frac{n}{n_i} \end{aligned} \quad (419)$$

where  $m_t$  is the carrier tunneling mass and  $E_{trap}$  is an energy of trap level that is taken from SRH recombination if the model is applied to the lifetimes ( $\tau$ ) or from trap equations if it is applied to cross sections ( $\sigma$ ).

## Chapter 16: Generation–Recombination

Shockley–Read–Hall Recombination

When quantization is active (see [Chapter 14 on page 348](#)), the classical density:

$$n_{\text{cl}} = n \exp \frac{-\Delta}{kT} \quad (420)$$

rather than the true density  $n$  enters [Equation 419](#).

The Hurkx model has only one parameter,  $m_t$ , the carrier tunneling mass, which can be specified in the parameter file for electrons and holes as follows:

```
HurkxTrapAssistedTunneling {  
    mt = <value>, <value>  
}
```

## Density Correction for Schenk and Hurkx Trap-Assisted Tunneling Models

In the original Schenk model [8], the density  $n$  in [Equation 395](#) is replaced by:

$$\tilde{n} = n \exp -\frac{\gamma_n |\nabla E_{F,n}| (E_t - E_0)}{kTF} \quad (421)$$

with  $E_0$  and  $E_t$  according to [Equation 411](#) and [Equation 412](#).  $p$  is replaced by an analogous expression.

The parameters  $\gamma_n = n/(n + n_{\text{ref}})$  and  $\gamma_p = p/(p + p_{\text{ref}})$  are close to one for significantly large densities and vanish for small densities, switching off the density correction and improving numeric robustness. The reference densities  $n_{\text{ref}}$  and  $p_{\text{ref}}$  are specified (in  $\text{cm}^{-3}$ ) by the `DenCorRef` parameter pair in the `TrapAssistedTunneling` parameter set. They default to  $10^3 \text{ cm}^{-3}$ .

Parameters for the Schenk model are accessible in the `TrapAssistedTunneling` parameter set. The default parameters implemented in Sentaurus Device are related to the gold acceptor level:  $E_{\text{trap}} = 0 \text{ eV}$ ,  $S = 3.5$ , and  $\hbar\omega_0 = 0.068 \text{ eV}$ .

`TrapAssistedTunneling` provides a parameter `MinField` (specified in  $\text{Vcm}^{-1}$ ) used for smoothing at small electric fields. A value of zero (the default) disables smoothing.

---

## Dynamic Nonlocal Path Trap-Assisted Tunneling

The local Schenk and Hurkx trap-assisted tunneling (TAT) models described in [SRH Field Enhancement on page 477](#) might not give accurate results at low temperatures or near abrupt junctions with a strong, nonuniform, electric field profile. Moreover, the local TAT models are not suitable for computing TAT in heterojunctions where the band edge is modulated by the material composition rather than the electric field.

In addition to the local Schenk and Hurkx TAT models, Sentaurus Device provides dynamic nonlocal path Schenk and Hurkx TAT models. These models take into account nonlocal TAT

## Chapter 16: Generation–Recombination

### Shockley–Read–Hall Recombination

processes with the WKB transmission coefficient based on the exact tunneling barrier. In these models, electrons and holes are captured in or emitted from the defect level at different locations by the phonon-assisted tunneling process. As a result, the position-dependent electron and hole recombination rates as well as the recombination rate at the defect level are all different. For each location and for each carrier type, the tunneling path is determined dynamically based on the energy band profile rather than predefined by the nonlocal mesh. Therefore, these models do not require user-specification of the nonlocal mesh.

#### Note:

The nonlocal path TAT models are not suitable for AC or noise analysis as nonlocal derivative terms in the Jacobian matrix are not taken into account. The lack of derivative terms can degrade convergence when the high-field saturation mobility model is switched on, or when the series resistance is defined at electrodes, or when there is a floating region.

## Recombination Rate

To compute the net electron recombination rate, the model dynamically searches for the tunneling path with the following assumptions:

- The tunneling path is a straight line with its direction equal to the gradient of the conduction band at the starting position.
- The tunneling energy is equal to the conduction band energy at the starting position and is equal to the defect level ( $E_T(x) = E_{\text{trap}}(x) + E_i(x)$ ) at the ending position.
- Electrons can be captured in or emitted from the defect level at any location between the starting position and the ending position of the tunneling path.
- When the tunneling path encounters Neumann boundaries or semiconductor–insulator interfaces, it undergoes specular reflection.

For a given tunneling path of length  $l$  that starts at  $x = 0$  and ends at  $x = l$ , electron capture and emission between the conduction band at  $x = 0$  and the defect level at any position  $x$  for  $0 \leq x \leq l$  is possible. As a result, the net electron recombination rate at  $x = 0$  from the conduction band due to the TAT process  $R_{\text{net}, n}^{\text{TAT}}$  involves integration along the path as follows:

$$R_{\text{net}, n}^{\text{TAT}} = C_n \int_0^l \frac{\Gamma_C(x, E_C(0))}{\tau_n(x)} \frac{T(0) + T(x)}{2\sqrt{T(0)T(x)}} \left[ \exp\left[\frac{E_{F,n}(0) - E_C(0)}{kT(0)}\right] f^p(x) - \exp\left[\frac{E_T(x) - E_C(0)}{kT(x)}\right] f^n(x) \right] dx \quad (422)$$

$$C_n = |\nabla E_C(0)| \frac{N_C(0)}{kT(0)} \exp\left[\frac{E_{F,n}(0) - E_C(0)}{kT(0)}\right] + 1^{-\alpha} \quad (423)$$

## Chapter 16: Generation–Recombination

Shockley–Read–Hall Recombination

$$\Gamma_C(x, \varepsilon) = W_C(x, \varepsilon) \exp \int_0^x -2 \kappa_C(r, \varepsilon) dr \quad (424)$$

where:

- $\tau_n$  is the electron lifetime.
- $\alpha = 0$  for Boltzmann statistics and  $\alpha = 1$  for Fermi–Dirac statistics.
- $f^n$  is the electron occupation probability at the defect level.
- $f^p$  is the hole occupation probability at the defect level.
- $\kappa_C$  is the magnitude of the imaginary wavevector obtained from the effective mass approximation or from the Franz two-band dispersion relation ([Equation 823](#)).
- $W_C(x, \varepsilon) = 1$  for the nonlocal Hurkx TAT model. For the nonlocal Schenk TAT model,  $W_C(x, \varepsilon)$  is modeled as:

$$W_C(x, \varepsilon) = \frac{\hbar [E_C(x) - E_C(0)] \exp(0.5) W[\varepsilon - E_T(x)]}{4x [E_C(x) - E_T(x)] \sqrt{\pi m_C k T(x) W[E_C(x) + kT(x)/2 - E_T(x)]}} \quad (425)$$

$$W(\varepsilon) = \frac{1}{\sqrt{\chi}} \exp \left( \frac{\varepsilon}{2kT} + \chi \frac{z}{l + \chi} \right)^l \quad (426)$$

where:

- $\chi = \sqrt{l^2 + z^2}$ .
- $l = \varepsilon/\hbar\omega_0$ .
- $z = S/\sinh(\hbar\omega_0/2kT)$ , where  $\hbar\omega_0$  is the phonon energy and  $S$  is the Huang–Rhys constant.

The net hole recombination rate can be obtained in a similar way. The quasistatic electron and hole occupation probabilities at the defect level can be determined by balancing the net electron capture rate and the net hole capture rate.

## Using the Dynamic Nonlocal Path Trap-Assisted Tunneling Model

The model is activated in the SRH option as follows:

```
Recombination( ...
    SRH( ...
        NonlocalPath(
            Lifetime = Schenk | Hurkx # Hurkx model is used by default
            Fermi | -Fermi          # use alpha=1 (alpha=0 by default)
            TwoBand | -TwoBand       # use Franz two-band dispersion
                                    # (off by default)
        )
    )
)
```

## Chapter 16: Generation–Recombination

Shockley–Read–Hall Recombination

```
)  
)
```

`Lifetime=Hurkx` (default) selects the nonlocal Hurkx model, while `Lifetime=Schenk` selects the nonlocal Schenk model. The `Fermi` option sets  $\alpha$  equal to 1 ( $\alpha = 0$  by default). The `TwoBand` option activates the Franz two-band dispersion ([Equation 823 on page 833](#)) instead of the effective mass approximation for the computation of the imaginary wavevector.

The nonlocal path TAT model introduces no additional model parameters. The same minority carrier lifetime for the SRH model is used in the nonlocal path TAT model. The doping- and temperature-dependent lifetime as well as the lifetime loaded from file can be specified together with the nonlocal path TAT model.

### Note:

The SRH field-enhancement model must not be used with the nonlocal path TAT model to avoid double-counting of the TAT process.

The nonlocal Hurkx model uses the tunneling mass specified in the `HurkxTrapAssistedTunneling` parameter set. The tunneling mass, the phonon energy, and the Huang–Rhys coupling constant for the nonlocal Schenk model are specified in the `TrapAssistedTunneling` parameter set.

The maximum length of the tunneling path is determined by the `MaxTunnelLength` parameter in the `Band2BandTunneling` parameter set.

You can consider electron TAT from Schottky contacts or Schottky metal–semiconductor interfaces using the nonlocal path TAT model. To consider the electron capture and emission from the Schottky contact, you must specify the parameter `TATNonlocalPathNC`, which represents the room temperature effective density-of-states ( $N_C(300K)$ ) for the Schottky contact in the electrode-specific `Physics` section. For example:

```
Physics (electrode = "source") { ...  
    TATNonlocalPathNC = 2.0e19  
}
```

Similarly, to consider the TAT from the Schottky metal–semiconductor interface, you must specify the room temperature effective density-of-states in the corresponding interface-specific `Physics` section. For example:

```
Physics (MaterialInterface = "Gold/Silicon") { ...  
    TATNonlocalPathNC = 2.0e19  
}
```

Sentaurus Device assumes that the effective density-of-states at  $T$  follows:

$$N_C(T) = N_C(300K) \frac{T^{\frac{3}{2}}}{300K} \quad (427)$$

## Chapter 16: Generation–Recombination

### Shockley–Read–Hall Recombination

Electron and hole recombination rates and the recombination rate at the defect level can be plotted as:

```
Plot { ...
    eSRHRecombination      # electron recombination rate
    hSRHRecombination      # hole recombination rate
    tSRHRecombination      # recombination rate at the defect level
}
```

---

## Trap-Assisted Auger Recombination Model

The trap-assisted Auger (TAA) recombination model is a modification of the SRH recombination and coupled defect level recombination models (see [Shockley–Read–Hall Recombination on page 473](#) and [Coupled Defect Level Recombination on page 487](#)).

When TAA is active, Sentaurus Device uses the lifetimes [4]:

$$\frac{\tau_p}{1 + \tau_p/\tau_p^{\text{TAA}}} \quad (428)$$

$$\frac{\tau_n}{1 + \tau_n/\tau_n^{\text{TAA}}} \quad (429)$$

instead of the lifetimes  $\tau_p$  and  $\tau_n$  in [Equation 395](#) and [Equation 436](#).

The TAA lifetimes in [Equation 428](#) depend on the carrier densities:

$$\frac{1}{\tau_n^{\text{TAA}}} \approx C_p^{\text{TAA}}(n + p) \quad (430)$$

$$\frac{1}{\tau_p^{\text{TAA}}} \approx C_n^{\text{TAA}}(n + p) \quad (431)$$

A reasonable order of magnitude for the TAA coefficients  $C_n^{\text{TAA}}$  and  $C_p^{\text{TAA}}$  is  $1 \times 10^{-12} \text{ cm}^3 \text{s}^{-1}$  to  $1 \times 10^{-11} \text{ cm}^3 \text{s}^{-1}$ ; the default values are  $C_n^{\text{TAA}} = C_p^{\text{TAA}} = 1 \times 10^{-12} \text{ cm}^3 \text{s}^{-1}$ .

TAA recombination is activated by using the keyword `TrapAssistedAuger` in the `Recombination` statement in the `Physics` section (see [Table 237 on page 1630](#)):

```
Physics {
    Recombination(TrapAssistedAuger ...)
    ...
}
```

The trap-assisted Auger parameters  $C_n^{\text{TAA}}$  and  $C_p^{\text{TAA}}$  are accessible in the parameter set `TrapAssistedAuger`.

## Chapter 16: Generation–Recombination

### Surface SRH Recombination Model

## Surface SRH Recombination Model

You can activate the surface SRH recombination model at semiconductor–semiconductor and semiconductor–insulator interfaces (see [Interface-Specific Models on page 68](#)).

At interfaces, an additional formula is used that is structurally equivalent to the bulk expression of the SRH generation–recombination:

$$R_{\text{surf, net}}^{\text{SRH}} = \frac{np - n_{i,\text{eff}}^2}{(n + n_1)/s_p + (p + p_1)/s_n} \quad (432)$$

with:

$$n_1 = n_{i,\text{eff}} \exp \left[ \frac{-E_{\text{trap}}}{kT} \right] \quad \text{and} \quad p_1 = n_{i,\text{eff}} \exp \left[ \frac{-E_{\text{trap}}}{kT} \right] \quad (433)$$

For Fermi statistics and quantization, the equations are modified in the same way as for bulk SRH recombination (see [Equation 399](#)).

The recombination velocities of otherwise identically prepared surfaces depend, in general, on the concentration of dopants at the surface [\[9\]\[10\]\[11\]](#).

Particularly, in cases where the doping concentration varies along an interface, including such a doping dependence is advised. Sentaurus Device models doping dependence of surface recombination velocities according to:

$$s = s_0 \left[ 1 + s_{\text{ref}} \left( \frac{N_i}{N_{\text{ref}}} \right)^\gamma \right] \quad (434)$$

The results of Cuevas [\[11\]](#) indicate that for phosphorus-diffused silicon,  $\gamma = 1$ . The results of King and Swanson [\[10\]](#) imply that no significant doping dependence exists for the recombination velocities of boron-diffused silicon surfaces.

To activate the model, specify the option `SurfacesRH` to the `Recombination` keyword in the `Physics` section for the respective interface. To plot the surface recombination, specify `SurfaceRecombination` in the `Plot` section. The parameters  $E_{\text{trap}}$ ,  $s_{\text{ref}}$ ,  $N_{\text{ref}}$ , and  $\gamma$  are accessible in the parameter set `SurfaceRecombination`.

*Table 84 Parameters of the surface SRH recombination model for silicon*

Symbol	Parameter name	Default value		Unit
		Electrons	Holes	
$s_0$	<code>S0</code>	$1 \times 10^3$	$1 \times 10^3$	cm/s
$s_{\text{ref}}$	<code>Sref</code>	$1 \times 10^{-3}$	1	

*Table 84 Parameters of the surface SRH recombination model for silicon (Continued)*

<b>Symbol</b>	<b>Parameter name</b>	<b>Default value</b>		<b>Unit</b>
		<b>Electrons</b>	<b>Holes</b>	
$N_{\text{ref}}$	Nref	$1 \times 10^{16}$		$\text{cm}^{-3}$
$\gamma$	gamma	1	1	
$E_{\text{trap}}$	Etrap	0		eV

The doping dependence of the recombination velocity can be suppressed by setting  $s_{\text{ref}}$  to zero.

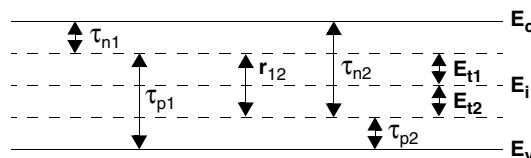
**Note:**

The SurfaceSRH keyword also is supported for metal–semiconductor interfaces, but the model activated there differs from the one described here (see [Electric Boundary Conditions for Metals on page 296](#)).

## Coupled Defect Level Recombination

The steady-state recombination rate for two coupled defect levels (CDLs) generalizes the familiar single-level SRH formula. An important feature of the model is a possibly increased field effect that might lead to large excess currents. The model is discussed in the literature [12]. Figure 20 illustrates the notation of the model parameters.

*Figure 20 Notation for CDL recombination including all capture and emission processes*



The CDL recombination rate is given by:

$$R = R_1 + R_2 + \sqrt{R_{12}^2 - S_{12}} - R_{12} \times \frac{\tau_{n1}\tau_{p2}(n + n_2)(p + p_1) - \tau_{n2}\tau_{p1}(n + n_1)(p + p_2)}{r_1 r_2} \quad (435)$$

with:

$$r_j = \tau_{nj}(p + p_j) + \tau_{pj}(n + n_j) \quad (436)$$

## Chapter 16: Generation–Recombination

### Radiative Recombination Model

$$R_j = \frac{np - n_{i,\text{eff}}^2}{r_j} , \quad j = 1, 2 \quad (437)$$

$$R_{12} = \frac{r_1 r_2}{2r_{12}\tau_{n1}\tau_{n2}\tau_{p1}\tau_{p2}(1-\varepsilon)} + \frac{\tau_{n1}(p+p_1) + \tau_{p2}(n+n_2)}{2\tau_{n1}\tau_{p2}(1-\varepsilon)} + \frac{\varepsilon[\tau_{n2}(p+p_2) + \tau_{p1}(n+n_1)]}{2\tau_{n2}\tau_{p1}(1-\varepsilon)} \quad (438)$$

$$S_{12} = \frac{1}{\tau_{n1}\tau_{p2}(1-\varepsilon)} \left( 1 - \frac{\tau_{n1}\tau_{p2}}{\tau_{n2}\tau_{p1}} \varepsilon \right) (np - n_{i,\text{eff}}^2) \quad (439)$$

$$\varepsilon = \exp \left( -\frac{|E_{t2} - E_{t1}|}{kT} \right) \quad (440)$$

where  $\tau_{ni}$  and  $\tau_{pi}$  denote the electron and hole lifetimes of the defect level  $i$ . The coupling parameter between the defect levels is called  $r_{12}$  (keyword `TrapTrapRate` in the parameter file). Carrier lifetimes are calculated analogously to carrier lifetimes in the SRH model (see [Shockley–Read–Hall Recombination on page 473](#)). The number of parameters is doubled compared to the SRH model, and you can change them in the `CDL` parameter set.

The quantities  $n_2$  and  $p_2$  are the corresponding quantities of  $n_1$  and  $p_1$  for the second defect level. They are defined analogously to [Equation 396](#) and [Equation 397](#).

For Fermi statistics and quantization, the equations are modified in the same way as for SRH recombination (see [Equation 399](#)).

## Using Coupled Defect Level Recombination

You can switch on CDL recombination using the keyword `CDL` in the `Physics` section:

```
Physics{ Recombination( CDL ... ) ... }
```

The contributions  $R_1$  and  $R_2$  in [Equation 437](#) can be plotted using the keywords `CDL1` and `CDL2` in the `Plot` section. For the net rate and coupling term,  $R - R_1 - R_2$ , the keywords `CDL` and `CDL3` must be specified.

## Radiative Recombination Model

The radiative (direct) recombination model expresses the recombination rate as:

$$R_{\text{net}} = C \cdot (np - n_{i,\text{eff}}^2) \quad (441)$$

By default, Sentaurus Device selects  $C = 2 \times 10^{-10} \text{ cm}^3 \text{s}^{-1}$  for GaAs and  $C = 0 \text{ cm}^3 \text{s}^{-1}$  for other materials. For Fermi statistics and quantization, the equations are modified in the same way as for SRH recombination (see [Equation 399](#)).

## Chapter 16: Generation–Recombination

### Auger Recombination Model

---

## Using the Radiative Recombination Model

The keyword `Radiative` in the `Physics` section activates the radiative recombination model:

```
Physics { Recombination (Radiative) }
```

The model can also be switched on or off using the notation `+Radiative` or `-Radiative`:

```
Physics (Region = "gate") {
    Recombination (+Radiative)
}
Physics (Material = "AlGaAs") {
    Recombination (-Radiative)
}
```

The value of the radiative recombination rate is plotted as follows:

```
Plot { RadiativeRecombination }
```

The value of the parameter  $C$  can be changed in the parameter file:

```
RadiativeRecombination { C = 2.5e-10 }
```

---

## Auger Recombination Model

The rate of band-to-band Auger recombination  $R_{\text{net}}^{\text{A}}$  is given by:

$$R_{\text{net}}^{\text{A}} = C_n n + C_p p \quad np - n_{\text{eff}}^2 \quad (442)$$

with temperature-dependent Auger coefficients [13][14][15]:

$$C_n(T) = A_{\text{A},n} + B_{\text{A},n} \left[ \frac{T}{T_0} + C_{\text{A},n} \left( \frac{T}{T_0} \right)^2 \left[ 1 + H_n \exp \left( -\frac{n}{N_{0,n}} \right) \right] \right] \quad (443)$$

$$C_p(T) = A_{\text{A},p} + B_{\text{A},p} \left[ \frac{T}{T_0} + C_{\text{A},p} \left( \frac{T}{T_0} \right)^2 \left[ 1 + H_p \exp \left( -\frac{p}{N_{0,p}} \right) \right] \right] \quad (444)$$

where  $T_0 = 300$  K. There is experimental evidence for a decrease of the Auger coefficients at high injection levels [15]. This effect is explained as resulting from exciton decay: at lower carrier densities, excitons, which are loosely bound electron–hole pairs, increase the probability for Auger recombination. Excitons decay at high carrier densities, resulting in a decrease of recombination. This effect is modeled by the terms  $1 + H \exp(-n/N_0)$  in Equation 443 and Equation 444.

Auger recombination is typically important at high carrier densities. Therefore, this injection dependence is seen only in devices where extrinsic recombination effects are extremely

## Chapter 16: Generation–Recombination

### Intrinsic Recombination Model for Silicon

low, such as high-efficiency silicon solar cells. The injection dependence of the Auger coefficient can be deactivated by setting  $H$  to zero in the parameter file.

For Fermi statistics and quantization, the equations are modified in the same way as for SRH recombination (see [Equation 399 on page 474](#)). [Table 85](#) lists the default values for silicon.

*Table 85 Coefficients of the Auger recombination model*

Symbol	$A_A$ [ $\text{cm}^6\text{s}^{-1}$ ]	$B_A$ [ $\text{cm}^6\text{s}^{-1}$ ]	$C_A$ [ $\text{cm}^6\text{s}^{-1}$ ]	$H$ [1]	$N_0$ [ $\text{cm}^{-3}$ ]
Parameter name	A	B	C	H	N0
Electrons	$6.7 \times 10^{-32}$	$2.45 \times 10^{-31}$	$-2.2 \times 10^{-32}$	3.46667	$1 \times 10^{18}$
Holes	$7.2 \times 10^{-32}$	$4.5 \times 10^{-33}$	$2.63 \times 10^{-32}$	8.25688	$1 \times 10^{18}$

## Using the Auger Recombination Model

The model is activated by the argument `Auger` in the `Recombination` statement:

```
Physics{ Recombination( Auger ... ) ... }
```

By default, Sentaurus Device uses [Equation 442](#) only if  $R_{\text{net}}^A$  is positive and replaces the value by zero if  $R_{\text{net}}^A$  is negative. To use [Equation 442](#) for negative values (that is, to allow for Auger generation of electron–hole pairs), use the `WithGeneration` option to the `Auger` keyword:

```
Physics { Recombination( Auger(WithGeneration) ... ) ... }
```

The Auger parameters are accessible in the `Auger` parameter set.

## Intrinsic Recombination Model for Silicon

With improvements in the quality of silicon bulk material, the technological progress of passivation techniques, and advanced characterization methods, an accurate description of the intrinsic recombination mechanisms of silicon became important to predict the performance of silicon-based devices. Richter *et al.* [\[16\]](#) developed a model to describe intrinsic recombination in crystalline silicon at 300 K covering a wide range of doping and carrier injection levels. The intrinsic recombination rate is calculated according to:

$$R_{\text{intrinsic}} = (np - n_{i,\text{eff}}^2)[C_{n0} \cdot g_{eeh}(n_0) \cdot n_0 + C_{p0} \cdot g_{ehh}(p_0) \cdot p_0 + C_{\Delta n} \cdot \Delta n^{\exp_{\Delta n}} + B_{\text{low}} \cdot B_{\text{rel}}(n, p)] \quad (445)$$

## Chapter 16: Generation–Recombination

### Intrinsic Recombination Model for Silicon

The first two terms in the brackets stand for the common Auger expression with Coulomb enhancement factors  $g_{eeh}$  and  $g_{ehh}$ , which are dominant in the low-injection condition.

$n, p$  describe the electron and hole carrier concentrations,  $n_{i,\text{eff}}$  is the effective intrinsic carrier density, and  $n_0, p_0$  denote the electron and hole concentration at thermal equilibrium (zero voltage applied), that is, for p-type doping:

$$p_0 = N_a - N_d \quad n_0 = \frac{n_{i,\text{eff}}^2}{p_0} \quad (446)$$

For n-type doping:

$$n_0 = N_d - N_a \quad p_0 = \frac{n_{i,\text{eff}}^2}{n_0} \quad (447)$$

$N_a$  is the total acceptor doping concentration, and  $N_d$  is the total donor doping concentration.

The Coulomb enhancement factors  $g_{eeh}$  and  $g_{ehh}$  account for the enhanced probability of finding an electron and a hole in the immediate vicinity of each other:

$$g_{eeh}(n_0) = 1 + C_{geeh} \left( 1 - \tanh \left[ \frac{n_0}{N_{0,\text{eeh}}} \right] \right)^{E_{geeh}} \quad (448)$$

$$g_{ehh}(p_0) = 1 + C_{gehh} \left( 1 - \tanh \left[ \frac{p_0}{N_{0,\text{ehh}}} \right] \right)^{E_{gehh}} \quad (449)$$

The third term in the brackets of [Equation 445](#) is proportional to the excess carrier density  $\Delta n$  and describes the Auger recombination for the high-injection condition.

The last term in the brackets of [Equation 445](#) describes the injection-dependent radiative recombination according to Altermatt *et al.* [17]:

$$B_{\text{rel}}(n, p) = b_{\min} + \frac{b_{\max} - b_{\min}}{1 + \frac{(n+p)}{b_1}^{b_2} + \frac{(n+p)}{b_3}^{b_4}} \quad (450)$$

with:

$$b_{\min} = r_{\max} + \frac{r_{\min} - r_{\max}}{1 + \frac{T}{r_f} \frac{r_f}{r_i}} \quad (451)$$

$$b_1 = 2 \left[ s_{\max} + \frac{s_{\min} - s_{\max}}{1 + \frac{T}{s_i} \frac{s_i}{s_2}} \right] \quad (452)$$

## Chapter 16: Generation–Recombination

### Intrinsic Recombination Model for Silicon

$$b_3 = 2 \left[ w_{\max} + \frac{w_{\min} - w_{\max}}{1 + \frac{T}{w_1^{b_2}}} \right]. \quad (453)$$

$T$  is the temperature.

#### Note:

For [Equation 450](#), Reference [17] notes the terms  $n + p/b_1$  and  $n + p/b_2$ , instead of  $(n + p)/b_1$  and  $(n + p)/b_2$ . In addition, a factor of 2 is missing in [Equation 452](#) and [Equation 453](#). This has been verified with the author and corrected in the implementation.

You can switch on this model by specifying `intrinsicRichter` in the `Recombination` section, which has been implemented as an installed PMI model.

#### Note:

To plot the spatial distribution of this recombination in the TDR file, add `PMIRecombination` as a plot variable.

The model parameters can be set in the Sentaurus Device parameter file section:

```
Material = "Silicon" {
    intrinsicRichter {
        blow=4.73e-15
        rmax=0.25
        ...
    }
}
```

The parameter name and default value for each constant in [Equation 445](#)–[Equation 453](#) are taken from [16] and [17], and are listed in [Table 86](#).

*Table 86 Parameters and default values for constants*

Constant	Parameter name	Default value	Unit
$B_{\text{low}}$	<code>blow</code>	4.73e-15	$\text{cm}^3/\text{s}$
$b_{\max}$	<code>bmax</code>	1	1
$b_2$	<code>b2</code>	0.54	1
$b_4$	<code>b4</code>	1.25	1
$r_{\max}$	<code>rmax</code>	0.2	1
$r_{\min}$	<code>rmin</code>	0	1

**Chapter 16: Generation–Recombination**  
Intrinsic Recombination Model for Silicon

*Table 86 Parameters and default values for constants (Continued)*

Constant	Parameter name	Default value	Unit
$r_1$	r1	320	K
$r_2$	r2	2.5	1
$s_{\max}$	smax	1.5e18	cm <sup>-3</sup>
$s_{\min}$	smin	1e7	cm <sup>-3</sup>
$s_1$	s1	550	K
$s_2$	s2	3	1
$w_{\max}$	wmax	4e18	cm <sup>-3</sup>
$w_{\min}$	wmin	1e9	cm <sup>-3</sup>
$w_1$	w1	365	K
$w_2$	w2	3.54	1
$C_{n0}$	cn0	2.5e-31	cm <sup>6</sup> /s
$C_{geeh}$	geehC	13	1
$N_{0, eeh}$	N0eeh	3.3e17	cm <sup>-3</sup>
$E_{geeh}$	geehE	0.66	1
$C_{p0}$	cp0	8.5e-32	cm <sup>6</sup> /s
$C_{gehh}$	gehhC	7.5	1
$N_{0, ehh}$	N0ehh	7e17	cm <sup>-3</sup>
$E_{gehh}$	gehhE	0.63	1
$C_{\Delta n}$	cDn	3e-29	cm <sup>6</sup> /s
$\exp_{\Delta n}$	expDn	0.92	1

## Constant Carrier Generation Model

The simplest generation model computes a constant carrier generation  $G_{\text{const}}$  and is activated in the `Physics` section as follows:

```
Physics {
    Recombination (
        ConstantCarrierGeneration (value = 1e10)    # [cm^-3 s^-1]
    )
}
```

Alternatively, the value of  $G_{\text{const}}$  can be specified in the parameter file:

```
ConstantCarrierGeneration {
    value = 1e10                      # [cm^-3 s^-1]
}
```

You can ramp the generation rate by specifying a corresponding `Goal` in a Quasistationary command:

```
Goal { Model = "ConstantCarrierGeneration"
    Parameter = "value"
    Value = 1e20
}
```

To visualize  $G_{\text{const}}$ , plot the value of `PMIRecombination`:

```
Plot {
    PMIRecombination
}
```

The model also can be specified regionwise or materialwise.

**Note:**

The constant carrier generation model is functionally equivalent to the constant optical generation model (see [Constant Optical Generation on page 657](#)).

---

## Avalanche Generation

Electron–hole pair production due to avalanche generation (impact ionization) requires a certain threshold field strength and the possibility of acceleration, that is, wide space charge regions. If the width of a space charge region is greater than the mean free path between two ionizing impacts, charge multiplication occurs, which can cause electrical breakdown. The reciprocal of the mean free path is called the ionization coefficient  $\alpha$ . With these coefficients for electrons and holes, the generation rate can be expressed as:

$$G_{ii} = \frac{1}{q}(\alpha_n |\vec{J}_n| + \alpha_p |\vec{J}_p|) \quad (454)$$

## Chapter 16: Generation–Recombination

### Avalanche Generation

where  $\vec{J}_n$  and  $\vec{J}_p$  are the electron and hole current density vectors described in [Introduction to Carrier Transport Models on page 238](#).

Sentaurus Device implements several models for the ionization coefficients: van Overstraeten – de Man, Okuto–Crowell, Lackner, University of Bologna, the new University of Bologna, and Hatakeyama.

Sentaurus Device allows you to select the appropriate driving force for the simulation, that is, the method used to compute the accelerating field. Choices include `GradQuasiFermi`, `Eparallel`, `CarrierTempDrive`, and `ElectricField` (see [Driving Force on page 509](#)).

---

## Using Avalanche Generation

Avalanche generation is switched on by using the keyword `Avalanche` in the `Recombination` statement in the `Physics` section. The keywords `eAvalanche` and `hAvalanche` specify separate models or driving forces for the electron and hole ionization coefficients, respectively.

The model is selected by using one of the keywords `vanOverstraeten`, `Okuto`, `Lackner`, `UniBo`, `UniBo2`, or `Hatakeyama`. The default model is `vanOverstraeten`.

The driving force is selected using one of the keywords `GradQuasiFermi`, `Eparallel`, `CarrierTempDrive`, or `ElectricField`. The default driving force is `GradQuasiFermi`.

The following example selects the default van Overstraeten – de Man model for the electron impact ionization process with a driving force derived from electron temperature, and selects the Okuto–Crowell model for holes using the default driving force based on `GradQuasiFermi`:

```
Physics {
    Recombination(eAvalanche(CarrierTempDrive) hAvalanche(Okuto)...)
}
```

To include a dependency on energy bandgap in the avalanche generation models, specify the keyword `BandgapDependence` as an argument to `Avalanche`, `eAvalanche`, or `hAvalanche` (see the model descriptions in the next sections). For example:

```
Physics {
    Recombination(Avalanche(Lackner BandgapDependence))
}
```

To plot the avalanche generation rate, specify `AvalancheGeneration` in the `Plot` section. To plot either of the two terms on the right-hand side of [Equation 454](#) separately, specify `eAvalanche` or `hAvalanche`. To plot  $\alpha_n$  or  $\alpha_p$ , specify `eAlphaAvalanche` or `hAlphaAvalanche`, respectively.

By default, the electron and hole current density vectors  $\vec{J}_n$  and  $\vec{J}_p$  in [Equation 454](#) are computed in a mesh element using the Scharfetter–Gummel approximation applied to each

## Chapter 16: Generation–Recombination

### Avalanche Generation

element edge (see [Discretization on page 1177](#)). This is consistent with the computation of  $\nabla \cdot J_n$  and  $\nabla \cdot J_p$  in [Equation 57](#). This approximation is a good one to describe structures where the current density is low along the direction of large electrostatic potential change, such as the channel area of a MOSFET, but such a condition of low current density in the direction of potential change might not be satisfied for power-device applications. As a result, the Scharfetter–Gummel approximation in the avalanche generation term might lead to a requirement to have a very fine mesh or might result in small steps to be taken at a voltage ramp.

To overcome these issues, you can use another approximation for  $\vec{J}_n$  and  $\vec{J}_p$  in [Equation 454](#), which is based on the representation of these current density vectors as  $\vec{J}_n = -q\mu_n n \nabla \Phi_n$  and  $\vec{J}_p = -q\mu_p p \nabla \Phi_p$  (see [Equation 60](#) and [Equation 61](#)). This approximation assumes that the carrier densities  $n$  and  $p$  are constant in elements and also makes the current density vectors constant in an element. Typically, you could see a small change in the breakdown voltage with this approximation, but it provides better convergence and stability properties.

To activate this alternative approximation, specify:

```
Math {
    AvalDensGradQF
}
```

The contribution that avalanche generation makes to the device equations is determined by multiplying the avalanche generation rate by the element-vertex control volumes (areas in two dimensions) computed by the `AverageBoxMethod` algorithm. However, as noted in [Truncated Obtuse Elements on page 1183](#), the total vertex control volume for an obtuse element is not the same as the total geometric volume for the element and can sometimes be several times larger. This can result in an exaggerated number of avalanche-generated carriers from such elements, which can contribute to premature breakdown. This might be the case if you observe that the breakdown voltage is much lower than expected, and the avalanche generation in the structure is confined to one or a few elements. To prevent this, an option is available to use truncated element-vertex volumes for avalanche calculations, where the total truncated vertex volume for an element is exactly equal to the geometric volume. To activate this option, specify:

```
Math {
    ElementVolumeAvalanche
}
```

Sometimes, mostly for 3D breakdown simulations, nearly flat elements can cause convergence and premature breakdown problems (due to singular numeric errors). Such nearly flat elements have a very small volume and excluding these elements does not affect breakdown voltage, but it helps to resolve problems with numeric singularity. To exclude such elements from the avalanche generation calculations, use the following keyword:

```
Math {
    AvalFlatElementExclusion = <float> #[0 to 90 degree]
}
```

## Chapter 16: Generation–Recombination

### Avalanche Generation

In two dimensions, this keyword excludes elements from contributing to avalanche generation if all the angles are less than the specified value or larger than the 180°-specified value. In three dimensions, this keyword excludes elements if the maximum vertex *volume angle* for the element is less than the specified value, where the vertex volume angle is computed as the arcsine of the unit-edge volume (scalar triple product of the unit vectors along the three element edges that meet at the vertex).

#### Note:

This keyword excludes only flat elements that, by definition, have poor angles at every vertex (very close to 0° or 180° in two dimensions). In three dimensions, for the case of a 90° angle between three edges of a vertex in an element (this case represents a good element), the vertex unit-edge volume is the maximum (equal to 1), and this makes the vertex volume angle equal to 90° as well. For a nearly flat element, both the unit-edge volume and the volume angle are nearly zero.

The value of `AvalFlatElementExclusion` must not exceed 1–2°. Otherwise, too many elements might be excluded from the avalanche generation, causing a shift in breakdown voltage to higher values.

To control and see the locations of nearly flat elements in a device structure, you can specify an option to plot the minimum and maximum volume angles for each element as follows:

```
Plot {  
    AvalFlatElementMin/Element  AvalFlatElementMax/Element  
}
```

---

## van Overstraeten – de Man Model

This model is based on the Chynoweth law [18]:

$$\alpha(F_{\text{ava}}) = \gamma a \exp -\frac{\gamma b}{F_{\text{ava}}} \quad (455)$$

with:

$$\gamma = \frac{\tanh \left[ \frac{\hbar \omega_{\text{op}}}{2kT_0} \right]}{\tanh \left[ \frac{\hbar \omega_{\text{op}}}{2kT} \right]} \quad (456)$$

The factor  $\gamma$  with the optical phonon energy  $\hbar \omega_{\text{op}}$  expresses the temperature dependence of the phonon gas against which carriers are accelerated. The coefficients  $a$ ,  $b$ , and  $\hbar \omega_{\text{op}}$ , as measured by van Overstraeten and de Man [19], apply to fields in the range from  $1.75 \times 10^3 \text{ Vcm}^{-1}$  to  $6 \times 10^3 \text{ Vcm}^{-1}$  and are listed in [Table 87](#).

## Chapter 16: Generation–Recombination

### Avalanche Generation

Different values for  $a$  and  $b$  are used for high and low electric fields. The values  $a(\text{low})$ ,  $b(\text{low})$  are used in the low field range up to  $E_0$ , and the values  $a(\text{high})$ ,  $b(\text{high})$  are used in the high field above  $E_0$ .

If `BandgapDependence` is specified as an argument to `Avalanche`, the coefficient  $b$  in [Equation 455](#) is replaced with:

$$b \rightarrow \frac{\beta E_g}{q\lambda} \quad (457)$$

where  $E_g$  is the energy bandgap,  $\lambda$  is the optical-phonon mean free path for the carrier, and  $\beta$  is a proportionality constant. [Table 87](#) lists the default values for  $\lambda$  and  $\beta$ .

You can adjust the parameters in the `vanOverstraetenDeMan` parameter set.

*Table 87 Parameters of van Overstraeten – de Man model ([Equation 455](#)) for silicon*

Symbol	Parameter name	Default value		Valid range of electric field	Unit
		Electrons	Holes		
$a$	$a(\text{low})$	$7.03 \times 10^5$	$1.582 \times 10^6$	$1.75 \times 10^5 \text{ Vcm}^{-1}$ to $E_0$	$\text{cm}^{-1}$
	$a(\text{high})$	$7.03 \times 10^5$	$6.71 \times 10^5$	$E_0$ to $6 \times 10^5 \text{ Vcm}^{-1}$	
$b$	$b(\text{low})$	$1.231 \times 10^6$	$2.036 \times 10^6$	$1.75 \times 10^5 \text{ Vcm}^{-1}$ to $E_0$	$\text{V/cm}$
	$b(\text{high})$	$1.231 \times 10^6$	$1.693 \times 10^6$	$E_0$ to $6 \times 10^5 \text{ Vcm}^{-1}$	
$E_0$	<code>E0</code>	$4 \times 10^5$	$4 \times 10^5$		$\text{V/cm}$
$\hbar\omega_{\text{op}}$	<code>hbarOmega</code>	0.063	0.063		$\text{eV}$
$\lambda$	<code>lambda</code>	$62 \times 10^{-8}$	$45 \times 10^{-8}$		$\text{cm}$
$\beta$	$\beta(\text{low})$	0.678925	0.815009	$1.75 \times 10^5 \text{ Vcm}^{-1}$ to $E_0$	1
	$\beta(\text{high})$	0.678925	0.677706	$E_0$ to $6 \times 10^5 \text{ Vcm}^{-1}$	

## Okuto–Crowell Model

Okuto and Crowell [20] suggested the empirical model:

$$\alpha(F_{\text{ava}}) = a \cdot 1 + c(T - T_0) F_{\text{ava}}^{\gamma} \exp\left[-\frac{b[1 + d(T - T_0)]}{F_{\text{ava}}}^{\delta}\right] \quad (458)$$

where  $T_0 = 300\text{ K}$  and [Table 88](#) lists the user-adjustable parameters with their default values for silicon. These values apply to the range of the electric field from  $10^5\text{ Vcm}^{-1}$  to  $10^6\text{ Vcm}^{-1}$ .

If `BandgapDependence` is specified as an argument to `Avalanche`, the coefficient  $b$  in [Equation 458](#) is replaced with:

$$b \rightarrow \frac{\beta E_g}{q\lambda} \quad (459)$$

where  $E_g$  is the energy bandgap,  $\lambda$  is the optical-phonon mean free path for the carrier, and  $\beta$  is a proportionality constant. [Table 88](#) lists the default values for  $\lambda$  and  $\beta$ .

You can adjust the parameters in the `Okuto` parameter set.

*Table 88 Parameters of Okuto–Crowell model ([Equation 458](#)) for silicon*

<b>Symbol</b>	<b>Parameter name</b>	<b>Default value</b>		<b>Unit</b>
		<b>Electrons</b>	<b>Holes</b>	
$a$	<code>a</code>	0.426	0.243	$\text{V}^{-1}$
$b$	<code>b</code>	$4.81 \times 10^5$	$6.53 \times 10^5$	$\text{V/cm}$
$c$	<code>c</code>	$3.05 \times 10^{-4}$	$5.35 \times 10^{-4}$	$\text{K}^{-1}$
$d$	<code>d</code>	$6.86 \times 10^{-4}$	$5.67 \times 10^{-4}$	$\text{K}^{-1}$
$\gamma$	<code>gamma</code>	1	1	1
$\delta$	<code>delta</code>	2	2	1
$\lambda$	<code>lambda</code>	$62 \times 10^{-8}$	$45 \times 10^{-8}$	$\text{cm}$
$\beta$	<code>beta</code>	0.265283	0.261395	1

## Lackner Model

Lackner [21] derived a pseudo-local ionization rate in the form of a modification to the Chynoweth law, assuming stationary conditions. The temperature-dependent factor  $\gamma$  was introduced to the original model:

$$\alpha_v(F_{ava}) = \frac{\gamma a_v}{Z} \exp -\frac{\gamma b_v}{F_{ava}} \quad \text{where } v = n, p \quad (460)$$

with:

$$Z = 1 + \frac{\gamma b_n}{F_{ava}} \exp -\frac{\gamma b_n}{F_{ava}} + \frac{\gamma b_p}{F_{ava}} \exp -\frac{\gamma b_p}{F_{ava}} \quad (461)$$

$$\gamma = \frac{\tanh \frac{\hbar \omega_{op}}{2kT_0}}{\tanh \frac{\hbar \omega_{op}}{2kT}} \quad (462)$$

The default values for  $a$ ,  $b$ , and  $\hbar\omega_{op}$  apply to silicon for the range of the electric field from  $10^5 \text{ Vcm}^{-1}$  to  $10^6 \text{ Vcm}^{-1}$ .

If BandgapDependence is specified as an argument to Avalanche, then  $b$  in Equation 460 and Equation 461 is replaced with:

$$b \rightarrow \frac{\beta E_g}{q\lambda} \quad (463)$$

where  $E_g$  is the energy bandgap,  $\lambda$  is the optical-phonon mean free path for the carrier, and  $\beta$  is a proportionality constant.

The model parameters are accessible in the Lackner parameter set.

*Table 89 Parameters of Lackner model (Equation 460) for silicon*

Symbol	Parameter name	Default value		Unit
		Electrons	Holes	
$a$	$a$	$1.316 \times 10^6$	$1.818 \times 10^6$	$\text{cm}^{-1}$
$b$	$b$	$1.474 \times 10^6$	$2.036 \times 10^6$	$\text{V/cm}$
$\hbar\omega_{op}$	$hbarOmega$	0.063	0.063	eV
$\lambda$	$lambda$	$62 \times 10^{-8}$	$45 \times 10^{-8}$	cm
$\beta$	$beta$	0.812945	0.815009	1

## University of Bologna Impact Ionization Model

This model was developed for an extended temperature range between 25°C and 400°C. It is based on impact ionization data generated by the Boltzmann solver HARM [22]. It covers a wide range of electric fields (50 kVcm<sup>-1</sup> to 600 kVcm<sup>-1</sup>) and temperatures (300 K to 700 K). The model is calibrated against impact ionization measurements [23][24] in the whole temperature range.

For an updated version of this model, see [New University of Bologna Impact Ionization Model on page 502](#).

The model reads:

$$\alpha(F_{\text{ava}}, T) = \frac{F_{\text{ava}}}{a(T) + b(T)\exp\left[\frac{d(T)}{F_{\text{ava}} + c(T)}\right]} \quad (464)$$

The temperature dependence of the model parameters, determined by fitting experimental data, reads (for electrons):

$$a(T) = a_0 + a_1 t^{a_2} \quad b(T) = b_0 \quad c(T) = c_0 + c_1 t + c_2 t^2 \quad d(T) = d_0 + d_1 t + d_2 t^2 \quad (465)$$

and for holes:

$$a(T) = a_0 + a_1 t \quad b(T) = b_0 \exp[b_1 t] \quad c(T) = c_0 t^{c_1} \quad d(T) = d_0 + d_1 t + d_2 t^2 \quad (466)$$

where  $t = T/1\text{ K}$ .

If `BandgapDependence` is specified as an argument to `Avalanche`, then  $d(T)$  is modified as follows:

$$d(T) \rightarrow \frac{\beta E_g}{q\lambda} / d_0 \quad (d_0 + d_1 t + d_2 t^2) \quad (467)$$

where  $E_g$  is the energy bandgap,  $\lambda$  is the optical-phonon mean free path for the carrier, and  $\beta$  is a proportionality constant.

The model parameters are accessible in the `UniBo` parameter set.

*Table 90 Parameters of University of Bologna impact ionization model for silicon*

<b>Symbol</b>	<b>Parameter name</b>	<b>Default value</b>		<b>Unit</b>
		<b>Electrons</b>	<b>Holes</b>	
$a_0$	ha0	4.3383	2.376	V
$a_1$	ha1	$-2.42 \times 10^{-12}$	$1.033 \times 10^{-2}$	V
$a_2$	ha2	4.1233	0	1

## Chapter 16: Generation–Recombination

### Avalanche Generation

Table 90 Parameters of University of Bologna impact ionization model for silicon

Symbol	Parameter name	Default value		Unit
		Electrons	Holes	
$b_0$	hb0	0.235	0.17714	V
$b_1$	hb1	0	$-2.178 \times 10^{-3}$	1
$c_0$	hc0	$1.6831 \times 10^4$	$9.47 \times 10^{-3}$	Vcm <sup>-1</sup>
$c_1$	hc1	4.3796	2.4924	Vcm <sup>-1</sup> , 1
$c_2$	hc2	0.13005	0	Vcm <sup>-1</sup> , 1
$d_0$	hd0	$1.2337 \times 10^6$	$1.4043 \times 10^6$	Vcm <sup>-1</sup>
$d_1$	hd1	$1.2039 \times 10^3$	$2.9744 \times 10^3$	Vcm <sup>-1</sup>
$d_2$	hd2	0.56703	1.4829	Vcm <sup>-1</sup>
$\lambda$	lambda	$62 \times 10^{-8}$	$45 \times 10^{-8}$	cm
$\beta$	beta	0.680414	0.562140	1

#### Note:

When you select the University of Bologna impact ionization model for both carriers, that is, Recombination(Avalanche(UniBo)), Sentaurus Device also activates Auger generation (see [Auger Recombination Model on page 489](#)).

Specify `Auger(-WithGeneration)` to deactivate this generation term if necessary.

---

## New University of Bologna Impact Ionization Model

The model described in [University of Bologna Impact Ionization Model on page 501](#) was developed further [25][26][27] to cover an extended temperature range between 25°C and 500°C (773 K). It is based on impact ionization data generated by the Boltzmann solver HARM [22] and is calibrated against specially designed impact ionization measurements [23][24]. It covers a wide range of electric fields.

The model reads:

$$\alpha(F_{\text{ava}}, T) = \frac{F_{\text{ava}}}{a(T) + b(T) \exp\left[\frac{d(T)}{F_{\text{ava}} + c(T)}\right]} \quad (468)$$

## Chapter 16: Generation–Recombination

### Avalanche Generation

where the coefficients  $a$ ,  $b$ ,  $c$ , and  $d$  are polynomials of  $T$ :

$$a(T) = \sum_{k=0}^3 a_k \frac{T}{1\text{K}}^k \quad b(T) = \sum_{k=0}^{10} b_k \frac{T}{1\text{K}}^k \quad (469)$$

$$c(T) = \sum_{k=0}^3 c_k \frac{T}{1\text{K}}^k \quad d(T) = \sum_{k=0}^3 d_k \frac{T}{1\text{K}}^k$$

If `BandgapDependence` is specified as an argument to `Avalanche`, then  $d(T)$  is modified as follows:

$$d(T) \rightarrow \frac{\beta E_g}{q\lambda} / d_0 \sum_{k=0}^3 d_k \frac{T}{1\text{K}}^k \quad (470)$$

where  $E_g$  is the energy bandgap,  $\lambda$  is the optical-phonon mean free path for the carrier, and  $\beta$  is a proportionality constant.

The model parameters are accessible in the `UniBo2` parameter set.

#### Note:

The name of the model is case sensitive. In both the command file and the parameter file, the exact capitalization of `UniBo2` must be used.

*Table 91 Parameters of new University of Bologna impact ionization model for silicon*

Symbol	Electrons		Holes		Unit
	Parameter	Default value	Parameter	Default value	
$a_0$	<code>a0_e</code>	4.65403	<code>a0_h</code>	2.26018	V
$a_1$	<code>a1_e</code>	$-8.76031 \times 10^{-3}$	<code>a1_h</code>	0.0134001	V
$a_2$	<code>a2_e</code>	$1.34037 \times 10^{-5}$	<code>a2_h</code>	$-5.87724 \times 10^{-6}$	V
$a_3$	<code>a3_e</code>	$-2.75108 \times 10^{-9}$	<code>a3_h</code>	$-1.14021 \times 10^{-9}$	V
$b_0$	<code>b0_e</code>	-0.128302	<code>b0_h</code>	0.058547	V
$b_1$	<code>b1_e</code>	$4.45552 \times 10^{-3}$	<code>b1_h</code>	$-1.95755 \times 10^{-4}$	V
$b_2$	<code>b2_e</code>	$-1.0866 \times 10^{-5}$	<code>b2_h</code>	$2.44357 \times 10^{-7}$	V
$b_3$	<code>b3_e</code>	$9.23119 \times 10^{-9}$	<code>b3_h</code>	$-1.33202 \times 10^{-10}$	V

## Chapter 16: Generation–Recombination

### Avalanche Generation

Table 91 Parameters of new University of Bologna impact ionization model for silicon

Symbol	Electrons		Holes		Unit
	Parameter	Default value	Parameter	Default value	
$b_4$	b4_e	$-1.82482 \times 10^{-12}$	b4_h	$2.68082 \times 10^{-14}$	V
$b_5$	b5_e	$-4.82689 \times 10^{-15}$	b5_h	0	V
$b_6$	b6_e	$1.09402 \times 10^{-17}$	b6_h	0	V
$b_7$	b7_e	$-1.24961 \times 10^{-20}$	b7_h	0	V
$b_8$	b8_e	$7.55584 \times 10^{-24}$	b8_h	0	V
$b_9$	b9_e	$-2.28615 \times 10^{-27}$	b9_h	0	V
$b_{10}$	b10_e	$2.73344 \times 10^{-31}$	b10_h	0	V
$c_0$	c0_e	$7.76221 \times 10^3$	c0_h	$1.95399 \times 10^4$	Vcm <sup>-1</sup>
$c_1$	c1_e	25.18888	c1_h	-104.441	Vcm <sup>-1</sup>
$c_2$	c2_e	$-1.37417 \times 10^{-3}$	c2_h	0.498768	Vcm <sup>-1</sup>
$c_3$	c3_e	$1.59525 \times 10^{-4}$	c3_h	0	Vcm <sup>-1</sup>
$d_0$	d0_e	$7.10481 \times 10^5$	d0_h	$2.07712 \times 10^6$	Vcm <sup>-1</sup>
$d_1$	d1_e	$3.98594 \times 10^3$	d1_h	993.153	Vcm <sup>-1</sup>
$d_2$	d2_e	-7.19956	d2_h	7.77769	Vcm <sup>-1</sup>
$d_3$	d3_e	$6.96431 \times 10^{-3}$	d3_h	0	Vcm <sup>-1</sup>
$\lambda$	lambda_e	$62 \times 10^{-8}$	lambda_h	$45 \times 10^{-8}$	cm
$\beta$	beta_e	0.391847	beta_h	0.831470	1

## Hatakeyama Avalanche Model

The Hatakeyama avalanche model [28] describes the anisotropic behavior in 4H-SiC power devices. The impact ionization coefficient  $\alpha$  is obtained according to the Chynoweth law [18]:

$$\alpha = \gamma a e^{-\frac{\gamma b}{F}} \quad (471)$$

with:

$$\gamma = \frac{\tanh \frac{\hbar \omega_{op}}{2kT_0}}{\tanh \frac{\hbar \omega_{op}}{2kT}} \quad (472)$$

The coefficients  $a$  and  $b$  are computed dependent on the direction of the driving force  $\vec{F}$ . For the hexagonal crystal coordinate system, the direction of anisotropy usually coincides with the (0001) lattice crystal direction, while the other crystal directions are considered to be isotropic, like (1120), for example. This allows for the decomposition of the driving force  $\vec{F}$  into two components, parallel and orthogonal to the direction of anisotropy, to account for avalanche anisotropy.

The norm  $F = \|F\|_2$  satisfies the equation:

$$F^2 = F_{0001}^2 + F_{11\bar{2}0}^2 \quad (473)$$

Based on the projections  $F_{0001}$  and  $F_{11\bar{2}0}$ , the coefficients  $a$  and  $b$  are computed as follows:

$$B = \frac{F}{\sqrt{F_{11\bar{2}0}^2 + F_{0001}^2}} \quad (474)$$

$$a = a_{11\bar{2}0} \frac{\frac{BF_{11\bar{2}0}}{b_{11\bar{2}0} F}}{a_{0001}} \quad (475)$$

$$A = \log \frac{a_{0001}}{a_{11\bar{2}0}} \quad (476)$$

$$b = B \sqrt{1 - \theta A^2 \frac{\frac{BF_{11\bar{2}0} F_{0001}}{F b_{11\bar{2}0} b_{0001}}^2} \quad (477)}$$

## Chapter 16: Generation–Recombination

### Avalanche Generation

With the default  $\theta = 1$ , coefficient  $b$  might become undefined for large values of the driving force  $F$  (the argument of the square root in [Equation 477](#) becomes negative). This only happens if:

$$F > \frac{2 \min(b_{0001}, b_{11\bar{2}0})}{\left| \log \frac{a_{0001}}{a_{11\bar{2}0}} \right|} \quad (478)$$

By setting  $\theta = 0$ , you have  $b = B$ , and coefficients  $a$  and  $b$  depend only on the direction of the driving force  $\vec{F}$ , not its magnitude.

*Table 92 Parameters for 4H-SiC coefficients of Hatakeyama avalanche model*

<b>Symbol</b>	<b>Parameter name</b>	<b>Default value</b>		<b>Unit</b>
		<b>Electrons</b>	<b>Holes</b>	
$a_{0001}$	a_0001	$1.76 \times 10^8$	$3.41 \times 10^8$	$\text{cm}^{-1}$
$a_{11\bar{2}0}$	a_1120	$2.10 \times 10^7$	$2.96 \times 10^7$	$\text{cm}^{-1}$
$b_{0001}$	b_0001	$3.30 \times 10^7$	$2.50 \times 10^7$	V/cm
$b_{11\bar{2}0}$	b_1120	$1.70 \times 10^7$	$1.60 \times 10^7$	V/cm
$\hbar\omega_{\text{op}}$	hbarOmega	0.19	0.19	eV
$\theta$	theta	1	1	1

The Hatakeyama avalanche model uses a special-purpose interpolation formula to compute the coefficients  $a$  and  $b$  based on the direction of the driving force. This formula differs from the standard approach as described in [Anisotropic Avalanche Generation on page 899](#).

### Driving Force of Hatakeyama Avalanche Model

In contrast to other avalanche models in Sentaurus Device (see [Driving Force on page 509](#)), which use only the magnitude  $F$  of the driving force, the Hatakeyama avalanche model requires a vectorial driving force  $\vec{F}$  to compute the coefficients  $a$  and  $b$ . Depending on the driving force model, the following expressions are used:

- **ElectricField:** The driving force  $\vec{F}$  is defined as the straight electric field  $\vec{E}$ .

## Chapter 16: Generation–Recombination

### Avalanche Generation

- Eparallel: The driving force  $\vec{F}$  is given by (where  $\vec{j}$  is the electron or hole current density):

$$\vec{F} = (\vec{E}, \vec{j}) \cdot \frac{\vec{j}}{\|\vec{j}\|^2} \quad (479)$$

- GradQuasiFermi: The driving force  $\vec{F}$  is given by (where  $\Phi$  is the electron or hole quasi-Fermi potential):

$$\vec{F} = (\nabla\Phi, \vec{j}) \cdot \frac{\vec{j}}{\|\vec{j}\|^2} \quad (480)$$

- CarrierTempDrive: The driving force  $\vec{F}$  is given by (where  $E^{\text{eff}}$  is the effective field obtained from the electron or hole carrier temperature):

$$\vec{F} = \vec{j} E^{\text{eff}} \quad (481)$$

See [Avalanche Generation With Hydrodynamic Transport on page 510](#).

## Default Anisotropic Coordinate System

For 2D simulations, Sentaurus Device assumes that the y-axis in the crystal coordinate system is the anisotropic axis 0001, and the x-axis in the crystal coordinate system is the isotropic axis 1120. For 3D simulations, the z-axis in the crystal coordinate system is the anisotropic axis, and the x-axis and y-axis span the isotropic plane. The simulation coordinate system relative to the crystal coordinate system is defined by the x and y vectors in the `LatticeParameters` section of the parameter file (see [Crystal and Simulation Coordinate Systems on page 886](#) and [Coordinate Systems on page 1005](#)).

## Specification of Anisotropic Direction

There are different possibilities to specifying the direction of anisotropy used by anisotropic models (in increasing priority):

1. Take the direction from the TDR file specification, coming from Sentaurus Process.
2. Specify the direction explicitly in the `LatticeParameters` section of a Sentaurus Device parameter file.
3. Use an explicit directional specification in the `Aniso` subsection of the Sentaurus Device command file.

## Chapter 16: Generation–Recombination

### Avalanche Generation

For avalanche models with an explicit anisotropic model parameter definition (such as Okuto–Crowell and van Overstraeten – de Man), specifying the anisotropic direction in the `Aniso` subsection must be assisted with the explicit activation of the anisotropic avalanche model:

```
Aniso (
    Avalanche
    Direction = (1 0 0)
)
```

For the Hatakeyama avalanche model, `Avalanche` is not required in the `Aniso` subsection because this model is implicitly anisotropic.

Previously, it was only possible to know the direction of anisotropy for the Hatakeyama avalanche model by looking at the `LatticeParameters` section in the parameter file. The explicit directional specification in the `Aniso` subsection was ignored. Moreover, if the `LatticeParameters` section came directly from Sentaurus Process, then the parameters were interpreted incorrectly in the 2D case.

Therefore, for the Hatakeyama avalanche model:

- If the `LatticeParameters` section comes directly from Sentaurus Process, then these parameters are interpreted correctly.
- Explicit specification of the direction of anisotropy in the command file is also supported for the Hatakeyama avalanche model.

#### Note:

The `-Hatakeyama` flag in the `Aniso` subsection allows for backward compatibility with previous releases.

### Examples

Assume you have a 2D problem with the Hatakeyama avalanche model. There is no `LatticeParameters` section in the parameter file, but the TDR file contains the following:

```
LatticeParameters {
    X = ( 0.0000e+00, 0.0000e+00, -1.0000e+00)
    Y = (-8.6603e-01, 0.5, 0.0000e+00)
}
```

Case 0: If the `Aniso` subsection of the `Physics` section has no direction specification, then the anisotropic direction is the z-axis of the crystal coordinate system, that is, the `x` and `y` vectors define the anisotropic direction along the `x`-axis of the simulation coordinate system.

```
Physics {
    # Case 1: Direct anisotropic direction specification. In this
    # case, Sentaurus Device does not use the X and Y vectors.
    Aniso( direction=(0 0 1) ) # z-axis of (default) crystal
                                # coordinate system
```

## Chapter 16: Generation–Recombination

### Avalanche Generation

```
# Case 2: Direct definition of anisotropic direction in simulation
# coordinate system.
Aniso( direction(SimulationSystem)=(1 0 0) )

# Case 3: Same as Case 2 - symbolic definition.
Aniso( direction(SimulationSystem)=xAxis )

# Case 4: Backward compatibility with Version N-2017.09.
Aniso(-Hatakeyama)
}
```

---

## Driving Force

In Sentaurus Device, the driving force  $F_{ava}$  for impact ionization can be computed as the component of the electrostatic field in the direction of the current,  $F_{ava} = \vec{F} \cdot \vec{J}_{n,p}$ , (keyword `Eparallel`), or the value of the gradient of the quasi-Fermi level,  $F_{ava} = |\nabla \Phi_{n,p}|$ , (keyword `GradQuasiFermi`). For these two possibilities,  $F_{ava}$  is affected by the keyword `ParallelToInterfaceInBoundaryLayer` (see [Field Correction Close to Interfaces on page 452](#)). For hydrodynamic simulations,  $F_{ava}$  can be computed from the carrier temperature (keyword `CarrierTempDrive`, see [Avalanche Generation With Hydrodynamic Transport on page 510](#)). The default model is `GradQuasiFermi` and, only for hydrodynamic simulations, it is `CarrierTempDrive`. See [Table 238 on page 1631](#) for a summary of keywords.

The option `ElectricField` is used to perform breakdown simulations using the *ionization integral* method (see [Approximate Breakdown Analysis on page 511](#)).

## Interpolation of Avalanche Driving Forces

As with high-field saturation mobility, Sentaurus Device provides interpolation of avalanche driving forces to the electric field at low carrier concentrations, which can sometimes improve convergence behavior.

For the `GradQuasiFermi` driving force, interpolation to the electric field is described in [Interpolation of the GradQuasiFermi Driving Force on page 451](#). As noted there, you can specify the following keywords for the interpolation parameters for electrons and holes in the `Math` section:

- `RefDens_eGradQuasiFermi_ElectricField`
- `RefDens_hGradQuasiFermi_ElectricField`

These keywords affect the calculation of the driving forces for *both* high-field saturation mobility and avalanche generation.

## Chapter 16: Generation–Recombination

### Avalanche Generation

For the Eparallel driving force, interpolation to the electric field is given by:

$$F_{ava,n} = \frac{n}{n+n_0} \vec{F} \cdot \hat{\vec{J}_n} + \frac{n_0}{n+n_0} \left| \vec{F} \right| \quad (482)$$

You can specify the reference densities for electrons and holes in the Math section by using the following parameters:

- RefDens\_eEparallel\_ElectricField\_Aval
- RefDens\_hEparallel\_ElectricField\_Aval

---

## Avalanche Generation With Hydrodynamic Transport

If the hydrodynamic transport model is used, the default driving force  $F_{ava}$  equals an effective field  $E^{\text{eff}}$  obtained from the carrier temperature.

The usual conversion of local carrier temperatures to effective fields  $E^{\text{eff}}$  is described by the algebraic equations:

$$n\mu_n E_n^{\text{eff}} = n \frac{3kT_n - T}{2q\lambda_n\tau_{en}} \quad (483)$$

$$p\mu_p E_p^{\text{eff}} = p \frac{3kT_p - T}{2q\lambda_p\tau_{ep}} \quad (484)$$

which are obtained from the energy conservation equation under time-independent, homogeneous conditions. [Equation 483](#) and [Equation 484](#) have been simplified in Sentaurus Device by using the assumption  $\mu_n E_n^{\text{eff}} = v_{\text{sat},n}$  and  $\mu_p E_p^{\text{eff}} = v_{\text{sat},p}$ . This assumption is true for high values of the electric field. However, for low field, the impact ionization rate is negligibly small.

The parameters  $\lambda_n$  and  $\lambda_p$  are fitting coefficients (default value 1) and their values can be changed in the parameter set `AvalancheFactors`, where they are represented as `n_1_f` and `p_1_f`, respectively.

The conventional conversion formulas [Equation 483](#) and [Equation 484](#) can be activated by specifying parameters  $\Upsilon_n = 0$ ,  $\Upsilon_p = 0$  in the same `AvalancheFactors` section.

The simplified conversion formulas predict a linear dependence of effective electric field on temperature for high values of carrier temperature. For silicon, however, Monte Carlo simulations do not confirm this behavior. To obtain a better agreement with Monte Carlo data, additional heat sinks must be taken into account by the inclusion of an additional term in the equations for  $E^{\text{eff}}$ .

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

Such heat sinks arise from nonelastic processes, such as impact ionization itself. Sentaurus Device supports the following model to account for these heat sinks:

$$n v_{\text{sat},n} E_n^{\text{eff}} = n \frac{3kT_n - T}{2q\lambda_n\tau_{\text{en}}} + \frac{\Upsilon_n}{q}(E_g + \delta_n kT_n) \alpha_n n v_{\text{sat},n} \quad (485)$$

A similar equation  $E_p^{\text{eff}}$  is used to determine  $E_p^{\text{eff}}$ . To activate this model, set the parameters  $\Upsilon_n$  and  $\Upsilon_p$  to 1. This is the default for silicon, where the generalized conversion formula [Equation 485](#) gives good agreement with Monte Carlo data for  $\delta_n = \delta_p = 3/2$ . For all other materials, the default of the parameters  $\Upsilon_n$  and  $\Upsilon_p$  is 0.

#### Note:

This procedure ensures that the same results are obtained as with the conventional local field-dependent models in the bulk case. Conversely, the temperature-dependent impact ionization model usually gives much more accurate predictions for the substrate current in short-channel MOS transistors.

*Table 93 Parameters of the hydrodynamic avalanche model*

Symbol	Parameter name	Default value
$\lambda_n$	n_1_f	1
$\lambda_p$	p_1_f	1
$\Upsilon_n$	n_gamma	1
$\Upsilon_p$	p_gamma	1
$\delta_n$	n_delta	1.5
$\delta_p$	p_delta	1.5

## Approximate Breakdown Analysis

Junction breakdown due to avalanche generation is simulated by inspecting the ionization integrals:

$$I_n = \alpha_n(x) e^{-\int_x^W (\alpha_n(x') - \alpha_p(x')) dx'} \quad (486)$$

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

$$I_p = \alpha_p(x) e^{-\int_0^x (\alpha_p(x') - \alpha_n(x')) dx'} \quad (487)$$

where  $\alpha_n$ ,  $\alpha_p$  are the ionization coefficients for electrons and holes, respectively, and  $W$  is the width of the depletion zone. The integrations are performed along field lines through the depletion zone. Avalanche breakdown occurs if both ionization integrals are greater than or equal to one. [Equation 486](#) describes electron injection (electron primary current) and [Equation 487](#) describes hole injection. Since these breakdown criteria do not depend on current densities, you can perform a breakdown analysis by computing only the Poisson equation and ionization integrals under the assumption of constant quasi-Fermi levels in the depletion region.

---

## Using Breakdown Analysis

To enable breakdown analysis, Sentaurus Device provides the driving force `ElectricField`, which can be computed even for constant quasi-Fermi levels.

### Note:

The driving force `ElectricField` is less physical than the others available in Sentaurus Device. Therefore, you should use it only for approximate breakdown analysis.

Specifying `ComputeIonizationIntegrals` in the `Math` section switches on computation of the ionization integrals. For example:

```
Math{
    ...
    ComputeIonizationIntegrals()
}
```

## Initial Element List

Depending on the details of the device design, it is generally not known which ionization path will contribute the most to breakdown. Sentaurus Device offers various strategies to generate the initial set of elements from which the path search starts. The following electric field-based restrictions of the initial set of elements are available:

- `ComputeAtMaxElectricField`:  
The path search starts from the element having the maximal electric field within a semiconductor region. This strategy is the fastest since only one path is evaluated at every bias point. However, there might be another path that contributes more to breakdown. In structures where the maximum electric field location jumps as a function of applied bias, this can cause nonsmooth curves for ionization integrals.

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

- `ComputeAtMaxElectricField(delta=<float>):`  
All elements within a semiconductor region where the electric field is higher than the maximal electric field within a semiconductor region minus `delta` are taken into the initial element list. The parameter `delta` is given in V/cm. This is the default option.
- `MinElectricField=<float>:`  
All elements within a semiconductor region where the electric field is higher than `MinElectricField`, in units of V/cm, are taken into the initial element list.

In addition to electric field-based restrictions, the initial set of elements can be restricted by the spatial position:

- Using `ComputeAll`, all elements within a semiconductor region are taken into the initial element list. You should combine this strategy with another spatial restriction strategy. Otherwise, excessive computation times might occur in structures with a large number of elements.
- In the `ComputeIonizationIntegrals` section, the `Window` section can be specified with different geometric shapes. Only elements with center positions within the specified geometric shapes are considered for the path search. You can specify the following geometric shapes:
  - `Cuboid(corner1=<vector> corner2=<vector>):`  
`corner1` and `corner2` are the diagonally opposite corners of the cuboid.
  - `Sphere(center=<vector>, radius=<float>):`  
`center` is the vector to the center of the sphere, and `radius` is the radius of the sphere.
  - `Cylinder(center1=<vector> center2=<vector> radius=<float>):`  
`center1` and `center2` are vectors to the circular bases of the cylinder, and `radius` is the radius of the bases of the cylinder.

For example:

```
ComputeIonizationIntegrals (
    ComputeAll
    Window (
        Cuboid(corner1=(-1 -1 1) corner2=(2 -2 2))           # in um
        Sphere(center=(2.5 3.5 -4.5) radius=0.5)            # in um
        Cylinder(center1=(5.1 6.2 7.3) center2=(0 0.1 0) radius=1.1)
                                                # in um
    )
)
```

### Path Reflection

By default, the path is reflected at the device boundary. Using `-PathReflection()`, the path search stops if the algorithm steps into a nonsemiconductor material or outside the device. The path search stops if the path comes close to a contact.

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

Additional options within the `PathReflection()` section are available:

- `ReflectAtMaterial=<material>:`  
The path is reflected at the interface to the defined materials. Elements inside the specified materials are excluded from the initial element list.
- `ReflectAtRegion=<region>:`  
The path is reflected at the interface to the defined regions. Elements inside the specified regions are excluded from the initial element list.
- `CloseToContact=<float>:`  
The path search algorithm stops when the distance between the path point and a contact is smaller than the specified value given in  $\mu\text{m}$  (default: 0.1  $\mu\text{m}$ ).

For example:

```
ComputeIonizationIntegrals (
    PathReflection(
        ReflectAtMaterial="Oxide"
        ReflectAtMaterial="PolySi"
        ReflectAtRegion="region1"
        ReflectAtRegion="region2"
        CloseToContact=0.01           # um
    )
)
```

## Periodic Boundary Conditions

You can apply periodic boundary conditions to the path search by using the `PeriodicBC` keyword. If Robin periodic boundary conditions are specified in the `Math` section (see [Robin PBC Approach on page 285](#)), then the parameters are taken from there. If mortar periodic boundary conditions are specified, then periodic boundary conditions for the path search are switched off. The Robin periodic boundary conditions can be overwritten. For example:

```
ComputeIonizationIntegrals(
    PeriodicBC(
        (Direction=0 Coordinates=(-1.0 2.0))  # periodic in x-direction
        (Direction=1 Coordinates=(0.5 12.0))   # periodic in y-direction
    )
)
```

The keyword `Direction=<int>` specifies the normal direction of the periodic boundary planes where `Direction=0 | 1 | 2` correspond to the x-, y-, and z-direction. The keyword `Coordinates=(<float> <float>)` specifies the left and right periodic boundary planes. The path search stops if the periodic translation ends outside of the device.

Auxiliary control is given with the keyword `MaxReflections=<int>`, which controls the maximum number of boundary reflections until the path search stops.

All elements with center positions outside of the specified periodic boundary condition planes are excluded from the initial element list.

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

#### Note:

Periodic boundary conditions can be used only in the path search mode without path reflection. Therefore, path reflection is switched off automatically if periodic boundary conditions are used.

### Electrostatic Stopping Criteria for Path Search

In addition, two modes are available to stop the path search:

- Electric field condition (default): The path search stops when the absolute value of the electric field is smaller than a threshold value, which is controlled with the keyword `PathSearchStopField=<float>` (default is 1000 V/cm).
- Electrostatic potential condition: The path search stops when the electrostatic potential is smaller than the minimum electrostatic potential  $\phi_{\min}$  or larger than the maximum electrostatic potential  $\phi_{\max}$ , which are given by  $\phi_{\min}^c = \phi_{\min} + \Delta\phi_{\min}$  and  $\phi_{\max}^c = \phi_{\max} - \Delta\phi_{\max}$  with the minimal value  $\phi_{\min}$  and the maximal value  $\phi_{\max}$  of the electrostatic potential at contacts, and the electrostatic potential shifts  $\Delta\phi_{\min}$  and  $\Delta\phi_{\max}$ .

You can control the electrostatic potential shifts by using the keyword `PathSearchStopDeltaPotential=<float> <float>`, where  $\Delta\phi_{\min}$  corresponds to the first value and  $\Delta\phi_{\max}$  corresponds to the second value.

By default, Sentaurus Device reports only the path with the largest value. With the addition of the keyword `WriteAll`, information about all the computed ionization paths, according to the selected strategy, are written to the log file.

### Stopping Criteria for Breakdown Simulations

The keyword `BreakAtIonIntegral` in the `Math` section terminates the quasistationary simulation when the largest electron and hole ionization integrals per path are greater than or equal to one.

The complete syntax of this keyword is `BreakAtIonIntegral(<number> <value>)` where a quasistationary simulation finishes if the `<number>` of electron and hole ionization integrals per path is greater than or equal to `<value>`. For example:

```
Math { BreakAtIonIntegral }
```

### Visualizing Breakdown Paths

This section discusses visualizing breakdown paths.

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

#### Visualization in Current Plot File

When you use `ComputeIonizationIntegrals`, the values of the path with the highest ionization integrals  $I_n$  and  $I_p$  ([Equation 486](#) and [Equation 487](#)) are saved automatically in the current plot file (\*.plt file) as `PhiElectron` and `PhiHole`, respectively.

#### Visualization in TDR File: Save Best Breakdown Path as Volume Dataset in Existing TDR File

The keywords `eIonIntegral`, `hIonIntegral`, and `MeanIonIntegral` in the `Plot` section specify the corresponding ionization integrals. The best path is saved as the volume dataset in the existing TDR file. For example:

```
Plot {  
    eIonIntegral  
    hIonIntegral  
    MeanIonIntegral  
}
```

#### Visualization in TDR File: Save Several Breakdown Paths as 1D Datasets in Separate TDR File

The keywords `SaveEIonIntegralPaths` and `SaveHIonIntegralPaths` in the `ComputeIonizationIntegrals` section store breakdown paths in a TDR file. For example:

```
Math {  
    ComputeIonizationIntegrals (   
        SaveEIonIntegralPaths = 1  
        SaveHIonIntegralPaths = 1  
    )  
}
```

By default, Sentaurus Device stores the paths that have the largest ionization integral values for electrons and holes. You can set the required number of paths to store by using `SaveEIonIntegralPaths=<int>` and `SaveHIonIntegralPaths=<int>`.

Sentaurus Device also stores the interpolated field variables, specified in the `Plot` section, along with the breakdown paths.

You specify the output TDR file for visualizing breakdown paths in the `File` section as follows:

```
File {  
    IonIntegral = "@ABA@"  
}
```

The keyword `IonIntegral` generates a TDR file containing breakdown paths and interpolated fields along the paths.

## Chapter 16: Generation–Recombination

### Approximate Breakdown Analysis

## Typical Command File of Sentaurus Device

```
Electrode {
    { name="anode" Voltage=0 }
    { name="cathode" Voltage=600 }
}
File {
    grid      = "@grid@"
    doping    = "@doping@"
    current   = "@plot@"
    output    = "@log@"
    plot      = "@data@"
    IonIntegral = "@ABA@"
}
Physics {
    Mobility (DopingDependence HighFieldSaturation)
    Recombination(SRH Auger Avalanche(ElectricField))
}
Solve {
    Quasistationary(
        InitialStep=0.02 MaxStep=0.01 MinStep=0.01
        Goal {name=cathode voltage=1000}
    )
    { poisson }
}
Math {
    Iterations=100
    BreakAtIonIntegral
    ComputeIonizationIntegrals(
        MinElectricField = 2.0e5
        SaveEIonIntegralPaths = 1
        SaveHIonIntegralPaths = 1
    )
}
Plot {
    eIonIntegral hIonIntegral MeanIonIntegral
    eDensity hDensity
    ElectricField/Vector
    eAlphaAvalanche hAlphaAvalanche
}
```

---

## Approximate Breakdown Analysis With Carriers

Sentaurus Device allows ionization integrals to be calculated when solving the electron and hole current continuity equations. In this case, however, impact ionization-generated carriers will be included self-consistently in the solution, and the benefits of performing an approximate breakdown analysis (faster simulations with fewer convergence issues) will be lost.

## Chapter 16: Generation–Recombination

### Avalanche Breakdown Probability

To prevent the self-consistent inclusion of impact ionization-generated carriers in the solution of the device equations, specify the option `AvalPostProcessing` in the `Math` section:

```
Math { AvalPostProcessing }
```

This option allows ionization integrals to be calculated even when solving for carriers, but retains the benefits of an approximate breakdown analysis.

## Using GradQuasiFermi as a Driving Force

The direction field is used for the geometric path search through the device. It can be specified in the `ComputeIonizationIntegrals()` section using `direction=<string>` as follows:

- `direction="ElectricField"` (default): The path search follows the electric vector field.
- `direction="eGradQuasiFermi"`: The path search follows the gradient of the electron quasi-Fermi energy.
- `direction="hGradQuasiFermi"`: The path search follows the gradient of the hole quasi-Fermi energy.

The direction field is used for the path search stopping criterion `PathSearchStopField`.

The ionization field, that is, the driving force for the impact ionization, is used for the computation of the ionization coefficients in [Equation 486](#) and [Equation 487](#).

For example:

```
Math {
    ComputeIonizationIntegrals(
        direction="eGradQuasiFermi"
    )
}
Physics {
    Recombination(
        Avalanche(GradQuasiFermi)
    )
}
```

---

## Avalanche Breakdown Probability

For the simulation of single-photon avalanche diodes, knowledge of the avalanche breakdown probability is useful. Sentaurus Device can compute the electron  $P_e(r)$ , the hole  $P_h(r)$ , and the joint  $P_j(r)$  avalanche breakdown probability according to the model of McIntyre [29].

## Chapter 16: Generation–Recombination

### Avalanche Breakdown Probability

The joint avalanche breakdown probability along the electric field line  $s(r)$  is given by:

$$P_j(s) = \frac{P_h(0)f(s)}{P_h(0)f(s) + 1 - P_h(0)} \quad (488)$$

where  $f(s) = \exp\left[-\int_0^s (\alpha_e - \alpha_h)ds'\right]$ . Sentaurus Device solves the following line integrals along electric field lines in the multiplication region:

$$-\log[1 - P_h(0)] = \int_0^w \alpha_h(s)P_j(s)ds \quad (489)$$

$$1 - P_h(s) = (1 - P_h(0))\exp\left[\int_0^s \alpha_h(s')P_j(s')ds'\right] \quad (490)$$

The multiplication region is defined as the region where the electric field is higher than a threshold value. Outside the multiplication region,  $P_e(r) = P_h(r) = P_j(r) = 0$ .

---

## Using Avalanche Breakdown Probability

The `BreakdownProbability` statement in the `Physics` section controls the computation of the avalanche breakdown probability:

```
Physics {...
    BreakdownProbability()
}
```

For a better approximation of carrier paths, you can specify the keyword `InterpolatedDiscretization` in the `BreakdownProbability` statement. You can control the threshold electric field for the definition of the multiplication region with the keyword `MinElectricField` (default: 1.0e5 V/cm). For example:

```
Physics {...
    BreakdownProbability(
        MinElectricField=1.0e4
        InterpolatedDiscretization
    )
}
```

The model for the impact ionization coefficients  $\alpha_e$  and  $\alpha_h$  is read from the `Avalanche` section (see [Avalanche Generation on page 494](#)).

#### Note:

This feature is designed for quasistationary stimulations. For meaningful breakdown probabilities, you must exclude impact ionization-generated carriers from the device equations. Therefore, additionally use the `AvalPostprocessing` option in the `Math` section (see [Approximate Breakdown Analysis With Carriers on page 517](#)).

## Chapter 16: Generation–Recombination

### High-Field Entrance Position and Time

To plot the electron, the hole, and the joint avalanche breakdown probability, specify `eBreakdownProbability`, `hBreakdownProbability`, and `jBreakdownProbability` in the Plot section.

---

## High-Field Entrance Position and Time

The charge collection probability can be defined as the probability that a carrier generated at position  $x$  outside the multiplication region will reach a point  $m$  (high-field entrance position) on the border of the multiplication region. Knowledge of the high-field entrance position is also useful for the computation of quantities that are related to the breakdown probability.

As APDs and SPADs are reverse biased devices, minority carrier trajectories starting from  $x$  and reaching  $m$  are of interest:

- The local net doping is n-type,  $n > p$ : The minority hole current vector  $J_h$  is followed.
- The local net doping is p-type,  $p \geq n$ : The negative minority electron current vector  $-J_e$  is followed.

---

## Using High-Field Entrance Position and Time

The keyword `ComputeCarrierPath` in the `Math` section controls the computation of the high-field entrance position. You can control the threshold electric field for the definition of the multiplication region with the keyword `thresholdField` (default: `1.0e5 V/cm`). For example:

```
Math {  
    ...  
    ComputeCarrierPath(thresholdField=2.0e4)  
}
```

The path search starts either from every vertex position or element center position inside a semiconductor material. This can be controlled with the keyword `startFrom="vertex"` (default) or `startFrom="element"`. For example:

```
ComputeCarrierPath(  
    startFrom = "element"  
)
```

The path search is reflected at device boundaries by default. By default, the path search stops as well when it reaches contacts. In addition, you can specify that the path search stops when it is close to a contact by using the keyword `closeToContact=<float>` (in  $\mu\text{m}$ ).

## Chapter 16: Generation–Recombination

### High-Field Entrance Position and Time

For example:

```
ComputeCarrierPath(  
    PathReflection(closeToContact=0.01)  
)
```

Results are saved in the following datasets:

- **HighFieldEntrancePosition**: A vector field where every position  $x$  is assigned to the high-field entrance position  $m$
- **HighFieldEntranceVertex**: A scalar field where every position  $x$  is assigned to the vertex that is the closest vertex from  $m$  inside the multiplication region
- **HighFieldEntranceMask**: A scalar field where additional information is stored in the mask value (see [Table 94](#))

*Table 94 Description of mask values*

Mask value	Assignment of position	Description
0	$x \rightarrow x$	<ul style="list-style-type: none"><li>• <math>x</math> is outside the multiplication region</li><li>• Path search does not run into multiplication region</li></ul>
1	$x \rightarrow m$	<ul style="list-style-type: none"><li>• <math>x</math> is outside the multiplication region</li><li>• Follow <math>\vec{J}_h</math> in n-doped semiconductor region</li><li>• Found <math>m</math> in multiplication region</li></ul>
2	$x \rightarrow m$	<ul style="list-style-type: none"><li>• <math>x</math> is outside the multiplication region</li><li>• Follow <math>-\vec{J}_e</math> in p-doped semiconductor region</li><li>• Found <math>m</math> in multiplication region</li></ul>
3	$x \rightarrow x$	<ul style="list-style-type: none"><li>• <math>x</math> is inside the multiplication region</li><li>• No need for path search</li></ul>
-1	$x \rightarrow x$	All other cases

- **HighFieldEntranceTime**: A scalar field that computes the transit time of the carriers from the start position  $s = 0$  until the high-field entrance position  $s = m$  according to  $t = \int_0^m ds/v(s)$ , where the data entries eVelocity or hVelocity are used for  $v(s)$ , respectively. The transit time is nonzero for paths that reach the multiplication region.

---

## Visualization in TDR Files

This section discusses visualization in TDR files.

### Save Volume Dataset in Existing TDR File

To plot the datasets for the vertex-wise search, specify `HighFieldEntrancePosition/Vector`, `HighFieldEntranceVertex`, and `HighFieldEntranceMask` in the `Plot` section.

To plot the datasets for the elementwise search, specify `HighFieldEntrancePosition/Element/Vector`, `HighFieldEntranceVertex/Element`, and `HighFieldEntranceMask/Element`.

**Note:**

The datasets are designed for data transfer. Therefore, plotting `HighFieldEntrancePosition/Vector` and `HighFieldEntranceVertex` is not necessarily meaningful.

### Save Carrier Paths as 1D Datasets in Separate TDR File

You specify a separate output TDR file name for visualizing carrier paths in the `File` section using `CarrierPath`. For example:

```
File {
    CarrierPath = "myFileName.tdr"
}
```

Plotting the carrier paths is controlled by the `SaveEPaths` and `SaveHPaths` statements in the `ComputeCarrierPath` section. For large numbers of available paths, it is useful to reduce the total number of written paths using `skip=<int>`, which specifies the number of paths omitted for plotting (default: `skip=0`). The first path to be plotted can be shifted using `offset=<int>` (default: `offset=0`). For example:

```
Math {
    ComputeCarrierPath (
        SaveEPaths (
            skip = 25
            offset = 13
        )
        SaveHPaths (
            skip = 10
            offset = 0
        )
    )
}
```

## Chapter 16: Generation–Recombination

### Band-to-Band Tunneling Models

Alternatively, a list of path index numbers can be specified and plotted. For example:

```
Math {
    ComputeCarrierPath (
        SaveEPaths (
            list = (12 17 43 512 831)
        )
        SaveHPaths (
            list = (121 82 55)
        )
    )
}
```

Sentaurus Device also stores the interpolated field variables, specified in the `Plot` section, along the carrier paths. For large numbers of carrier paths and plot variables, you can reduce the loading time in Sentaurus Visual by switching off `withPlotData`. For example:

```
Math {
    ComputeCarrierPath (
        SaveEPaths ( -withPlotData )
    )
}
```

#### Note:

Dynamic datasets that are triggered from the following plot names are *not* stored in the path:

`AbsorbedPhotonDensity`, `ComplexRefractiveIndex`, `OpticalAbsorptionHeat`,  
`OpticalGeneration`, `OpticalIntensity`, `QuantumYield`, `QW_OverlapIntegral`,  
`ThermalizationYield`

---

## Band-to-Band Tunneling Models

Sentaurus Device provides different band-to-band tunneling models:

- [Schenk Band-to-Band Tunneling Model on page 526](#)
- [Hurkx Band-to-Band Tunneling Model on page 527](#)
- [Modified Hurkx Band-to-Band Tunneling Model on page 528](#)
- [Simple Band-to-Band Tunneling Models on page 529](#)
- [Dynamic Nonlocal Path Band-to-Band Tunneling Model on page 529](#)

The Schenk, Hurkx, modified Hurkx, and simple models use a common approach to suppress artificial band-to-band tunneling near insulator interfaces and for rapidly varying fields (see [Tunneling Near Interfaces and Equilibrium Regions on page 525](#)).

## Using Band-to-Band Tunneling

Band-to-band tunneling is controlled by the `Band2Band` option of `Recombination`:

```
Recombination( ...
    Band2Band (
        Model = Schenk | Hurkx | modifiedHurkx | E1 | E1_5 | E2 |
            NonlocalPath
        DensityCorrection = Local | None
        InterfaceReflection | -InterfaceReflection
        FranzDispersion | -FranzDispersion
        ParameterSetName = (<string>...)
    )
)
```

The following options for `Model` are available:

- `Schenk` selects the Schenk model.
- `Hurkx` selects the Hurkx model.
- `modifiedHurkx` selects the modified Hurkx model.
- `E1`, `E1_5`, and `E2` select one of the simple models.
- `NonlocalPath` selects the nonlocal path model.

The keyword `DensityCorrection` is used by the Schenk, Hurkx, modified Hurkx, and simple models, and its default value is `None`. A value of `Local` activates a local-density correction (see [Schenk Density Correction on page 527](#)).

The option `InterfaceReflection` is used by the nonlocal path model and is switched on by default. This option allows a tunneling path reflected at semiconductor–insulator interfaces. When it is switched off, band-to-band tunneling is neglected when the tunneling path encounters semiconductor–insulator interfaces.

The option `FranzDispersion` is used by the nonlocal path model and is switched off by default. When it is switched on, the Franz dispersion relation ([Equation 823](#)) instead of the Kane dispersion relation ([Equation 500](#)) for the imaginary wavevector is used in the direct tunneling process. The indirect tunneling process is not affected by this option.

All models use `ParameterSetName` to specify a list of names of `Band2BandTunneling` parameter sets. For each name, band-to-band tunneling is computed using the parameters in the named parameter set, and the results are all added to give the total band-to-band tunneling rate. Without `ParameterSetName`, only band-to-band tunneling obtained with the unnamed `Band2BandTunneling` parameter set is computed.

The following example uses the nonlocal path model, summing the contributions obtained with the `phonon-assisted` and `direct` parameter sets:

```
Band2Band(
    Model=NonlocalPath
    ParameterSetName=( "phonon-assisted" "direct" )
)
```

For backward compatibility:

- `Band2Band` (no parameters) selects the Schenk model with local-density correction.
- `Band2Band(Hurkx)` selects the Hurkx model without density correction.
- `Band2Band(E1)`, `Band2Band(E1_5)`, and `Band2Band(E2)` select one of the simple models.

The parameters for all band-to-band tunneling models are available in the parameter set `Band2BandTunneling`. The parameters specific to individual models are described in the respective sections. The parameter `MinField` (specified in  $V\text{cm}^{-1}$ ) is used by the Schenk, Hurkx, modified Hurkx, and simple models for smoothing at small electric fields. A value of zero (the default) deactivates smoothing.

Named `Band2BandTunneling` parameter sets are specified in the parameter file. The following example specifies a parameter set named `phonon-assisted`:

```
Band2BandTunneling "phonon-assisted" { ... }
```

**Note:**

Parameters in named `Band2BandTunneling` parameter sets do not have default values. Therefore, you must specify values for all the parameters used by the model you select in the command file.

## Tunneling Near Interfaces and Equilibrium Regions

Physically, band-to-band tunneling occurs over a certain tunneling distance. If the material properties or the electric field change significantly over this distance, [Equation 491](#), [Equation 494](#), and [Equation 498](#) become inaccurate. In particular, near insulator interfaces, band-to-band tunneling vanishes, as no states to tunnel to are available in the insulator.

In some parts of the device (near equilibrium regions), it is possible that the electric field is large but changes rapidly, so that the actual tunneling distance (the distance over which the electrostatic potential change amounts to the band gap) is bigger and, therefore, tunneling is much smaller than expected from the local field alone.

To account for these effects, additional control parameters are introduced in the parameter set `Band2BandTunneling`:

```
dDist = <value> # [cm]
dPot = <value> # [V]
```

By default, both parameters equal zero. Sentaurus Device deactivates band-to-band tunneling within a distance `dDist` from insulator interfaces. If `dPot` is nonzero (reasonable values for `dPot` are of the order of the band gap), Sentaurus Device deactivates band-to-band tunneling at each point where in the field direction, the change of the electrostatic potential over a distance of  $dPot/F$  is smaller than  $dPot/2$ .

## Schenk Band-to-Band Tunneling Model

Phonon-assisted band-to-band tunneling cannot be neglected in steep p-n junctions (with a doping level of  $1 \times 10^{19} \text{ cm}^{-3}$  or more on both sides) or in high normal electric fields of MOS structures. It must be switched on if the field, in some regions of the device, exceeds (approximately)  $8 \times 10^5 \text{ V/cm}$ . In this case, defect-assisted tunneling must also be switched on (see [SRH Field Enhancement on page 477](#)).

Band-to-band tunneling is modeled using the expression [30]:

$$R_{\text{net}}^{\text{bb}} = AF^{7/2} \frac{\tilde{n}\tilde{p} - n_{i,\text{eff}}^2}{(\tilde{n} + n_{i,\text{eff}})(p + n_{i,\text{eff}})} \left[ \frac{(F_C^\mp)^{-3/2} \exp -\frac{F_C^\mp}{F}}{\exp \frac{\hbar\omega}{kT} - 1} + \frac{(F_C^\pm)^{-3/2} \exp -\frac{F_C^\pm}{F}}{1 - \exp -\frac{\hbar\omega}{kT}} \right] \quad (491)$$

where  $\tilde{n}$  and  $\tilde{p}$  equal  $n$  and  $p$  for `DensityCorrection=None`, and are given by [Equation 493](#) for `DensityCorrection=Local`.

The critical field strengths read:

$$F_C^\pm = B(E_{g,\text{eff}} \pm \hbar\omega)^{3/2} \quad (492)$$

The upper sign in [Equation 491](#) refers to tunneling generation ( $np < n_{i,\text{eff}}^2$ ) and the lower sign refers to recombination ( $np > n_{i,\text{eff}}^2$ ). The quantity  $\hbar\omega$  denotes the energy of the transverse acoustic phonon.

For Fermi statistics and quantization, [Equation 491](#) is modified in the same way as for SRH recombination (see [Equation 399 on page 474](#)).

The model parameters [30] can be accessed in the `Band2BandTunneling` parameter set. The default values were obtained assuming the field direction to be `111`.

Table 95 Parameters of the Schenk band-to-band tunneling model

Symbol	Parameter name	Default value	Unit
A	A	$8.977 \times 10^{20}$	$\text{cm}^{-1}\text{s}^{-1}\text{V}^{-2}$
B	B	$2.14667 \times 10^7$	$\text{V}\text{cm}^{-1}\text{eV}^{-3/2}$
$\hbar\omega$	hbarOmega	18.6	meV

## Schenk Density Correction

The modified electron density reads:

$$\tilde{n} = n \frac{\gamma_n |\nabla E_{F,n}|}{N_C} \cdot \frac{F}{\text{cm}^2} \quad (493)$$

There is a similar relation for  $\tilde{p}$ . The parameters  $\gamma_n = n/(n + n_{\text{ref}})$  and  $\gamma_p = p/(p + p_{\text{ref}})$  work as discussed in [Density Correction for Schenk and Hurkx Trap-Assisted Tunneling Models on page 481](#). The reference densities  $n_{\text{ref}}$  and  $p_{\text{ref}}$  are specified (in  $\text{cm}^{-3}$ ) by the DenCorRef parameter pair in the Band2BandTunneling parameter set.

An additional parameter MinGradQF (specified in  $\text{eV cm}^{-1}$ ) is available for smoothing at small values of the gradient for the Fermi potential.

## Hurkx Band-to-Band Tunneling Model

Similar to the other band-to-band tunneling models, in the Hurkx model [31], tunneling carriers are modeled by an additional generation–recombination process. Its contribution is expressed as:

$$R_{\text{net}}^{\text{bb}} = A \cdot D \cdot \frac{F}{1 \text{ V/cm}} \exp \left( -\frac{BE_g(T)^{3/2}}{E_g(300\text{K})^{3/2}F} \right) \quad (494)$$

where:

$$D = \frac{np - n_{\text{i,eff}}^2}{(n + n_{\text{i,eff}})(p + n_{\text{i,eff}})} (1 - |\alpha|) + \alpha \quad (495)$$

Here, specifying  $\alpha = 0$  gives the original Hurkx model, whereas  $\alpha = -1$  gives only generation ( $D = -1$ ), and  $\alpha = 1$  gives only recombination ( $D = 1$ ). Therefore, if  $D < 0$ , it is a net carrier generation model. If  $D > 0$ , it is a recombination model. For Fermi statistics and quantization, [Equation 495](#) is modified in the same way as for SRH recombination (see [Equation 399](#)).

For `DensityCorrection=Local`,  $n$  and  $p$  in [Equation 495](#) are replaced by  $\tilde{n}$  and  $\tilde{p}$  (see [Schenk Density Correction on page 527](#)).

You can specify the coefficients  $A$  (in  $\text{cm}^{-3}\text{s}^{-1}$ ),  $B$  (in  $\text{V}/\text{cm}$ ),  $P$ , and  $\alpha$  in the parameter set `Band2BandTunneling`. By default, Sentaurus Device uses  $\alpha = 0$  and the parameters from the `E2` model (see [Table 96 on page 529](#)). Different values for the generation (`Agen`, `Bgen`, and `Pgen`) and recombination (`Arec`, `Brec`, and `Prec`) of carriers are supported.

For example, to change the parameters to those used in [\[31\]](#), use:

```
Band2BandTunneling {
    Agen = 4e14 # [1/(cm3s)]
    Bgen = 1.9e7 # [V/cm]
    Pgen = 2.5 # [1]
    Arec = 4e14 # [1/(cm3s)]
    Brec = 1.9e7 # [V/cm]
    Prec = 2.5 # [1]
    alpha = 0 # [1]
}
```

## Modified Hurkx Band-to-Band Tunneling Model

Due to the nonlocal nature of the dynamic nonlocal path band-to-band tunneling model, convergence problems might arise for some devices. Therefore, Sentaurus Device offers the modified Hurkx band-to-band tunneling model that resembles more closely the results from the dynamic nonlocal path band-to-band tunneling model and shows better convergence behavior due to its local nature.

[Equation 494](#) is modified according to:

$$R_{\text{net}}^{\text{mH}} = R_{\text{net}}^{\text{bb}} \cdot g \quad (496)$$

It differs from the Hurkx band-to-band tunneling model by a factor of:

$$g = \begin{cases} \frac{F - F_0}{F_0}^Q & \text{for } F \geq F_0 \\ 0 & \text{for } F < F_0 \end{cases} \quad (497)$$

where:

- $g$  is introduced to avoid large tunneling generation at zero bias with a fitting parameter  $Q$ .
- The local parameter  $F_0 = \max \tilde{F}_0, \sqrt{C \frac{2qE_g(300K)N_{\text{net}}}{\epsilon}}$ , with the equilibrium electric field at zero bias  $\tilde{F}_0$  and the scaling factor  $C$ .

## Chapter 16: Generation–Recombination

### Band-to-Band Tunneling Models

The coefficients  $Q$  and  $C$  can be specified in the Band2BandTunneling parameter set. By default, Sentaurus Device uses  $Q = 1.5$ ,  $C = 1$ , and  $\alpha = -1$ . For example, to change the parameters for this model, specify:

```
Band2BandTunneling {  
    Qgen = 1.5 # [1]  
    Cgen = 1.0 # [1]  
    Qrec = 2.5 # [1]  
    Crec = 1.0 # [1]  
}
```

---

## Simple Band-to-Band Tunneling Models

Sentaurus Device provides a family of simple band-to-band tunneling models. Compared to the advanced models, the greatest weakness of the simple models is that they predict a nonzero generation rate even in equilibrium.

A general expression for these models can be written for the generation term [32] as:

$$G^{\text{b2b}} = AF^P \exp \frac{-B}{F} \quad (498)$$

Depending on the value of Model,  $P$  takes the value 1, 1.5, or 2.

Table 96 lists the coefficients of the models and their default values. The coefficients  $A$  and  $B$  can be changed in the Band2BandTunneling parameter set.

Table 96     Coefficients of the simple band-to-band tunneling models

Model	P	A	B
E1	1	$1.1 \times 10^{27} \text{ cm}^{-2} \text{ s}^{-1} \text{ V}^{-1}$	$21.3 \times 10^6 \text{ Vcm}^{-1}$
E1_5	1.5	$1.9 \times 10^{24} \text{ cm}^{-1.5} \text{ s}^{-1} \text{ V}^{-1.5}$	$21.9 \times 10^6 \text{ Vcm}^{-1}$
E2	2	$3.4 \times 10^{21} \text{ cm}^{-1} \text{ s}^{-1} \text{ V}^{-2}$	$22.6 \times 10^6 \text{ Vcm}^{-1}$

---

## Dynamic Nonlocal Path Band-to-Band Tunneling Model

Sufficient band-bending caused by electric fields or heterostructures can make electrons in the valence band valley ( $\Gamma$ -valley in the  $k$ -space), at a certain location, reach the conduction band valley ( $\Gamma$ -,  $X$ -, or  $L$ -valley in the  $k$ -space) at different locations using direct or phonon-assisted band-to-band tunneling processes.

This model implements the nonlocal generation of electrons and holes caused by direct and phonon-assisted band-to-band tunneling processes [33]. In direct semiconductors such as

## Chapter 16: Generation–Recombination

### Band-to-Band Tunneling Models

GaAs and InAs, the direct tunneling process is usually dominant. On the other hand, the phonon-assisted tunneling process is dominant in indirect semiconductors such as Si and Ge. If energy differences between the conduction band valleys are small, it is possible that both the direct and the phonon-assisted tunneling processes are important.

The generation rate is obtained from the nonlocal path integration, and electrons and holes are generated nonlocally at the ends of the tunneling path. As a result, the position-dependent electron and hole generation rates are different in this model. The model can be applied to heterostructure devices with abrupt and graded heterojunctions.

The main difference between this model and the band-to-band tunneling model based on the nonlocal mesh (see [Band-to-Band Contributions to Nonlocal Tunneling Current on page 840](#)) is that the tunneling path is determined dynamically based on the energy band profile rather than predefined by the nonlocal mesh. Therefore, this model does not require user-specification of the nonlocal mesh.

The model dynamically searches for the tunneling path with the following assumptions:

- The tunneling path starts from the valence band minus the valence band offset in a region where the nonlocal path model is active.
- The tunneling path is a straight line with its direction opposite to the gradient of the valence band at the starting position. The valence band offset does not change the direction of the tunneling path.
- The tunneling energy is equal to the valence band energy minus the valence band offset at the starting position and is equal to the conduction band energy plus the conduction band offset at the ending position.
- When the tunneling path encounters Neumann boundaries or semiconductor–insulator interfaces, it undergoes specular reflection.
- The tunneling path ends at the conduction band plus the conduction band offset.

If the path crosses a region where the nonlocal path model is not active, by default, the tunneling from this path is discarded. If the `-eB2BGenWithinSelectedRegions` flag is specified in the global `Math` section, tunneling for all paths entirely within semiconductor regions are accounted for.

#### Note:

By default, nonlocal derivative terms in the Jacobian matrix are not taken into account. To use AC or noise analysis with the present model, computation of nonlocal derivatives must be switched on (see [Using the Nonlocal Path Band-to-Band Tunneling Model on page 533](#)). The lack of derivative terms can degrade convergence when the high-field saturation mobility model is switched on or the series resistance is defined at electrodes.

## Band-to-Band Generation Rate

For a given tunneling path of length  $l$  that starts at  $x = 0$  and ends at  $x = l$ , holes are generated at  $x = 0$  and electrons are generated at  $x = l$ .

The net hole recombination rate at  $x = 0$  due to the direct band-to-band tunneling process  $R_{\text{net}}^{\text{d}}$  can be written as:

$$R_{\text{net}}^{\text{d}} = |\nabla E_V(0)| C_d \exp -2 \int_0^l \kappa dx \left[ \exp \left[ \frac{\epsilon - E_{F,n}(l)}{kT(l)} \right] + 1 \right]^{-1} - \left[ \exp \left[ \frac{\epsilon - E_{F,p}(0)}{kT(0)} \right] + 1 \right]^{-1} \quad (499)$$

where:

- $C_d = \frac{g\pi}{36h} \int_0^l \frac{dx}{\kappa} \left[ 1 - \exp -k_m^2 \int_0^l \frac{dx}{\kappa} \right]$ .
- $h$  is Planck's constant.
- $g$  is the degeneracy factor.
- $\epsilon = E_V(0) - \Delta_V(0) = E_C(l) + \Delta_C(l)$  is the tunneling energy.
- $\Delta_C$  is the conduction band offset.  $\Delta_C$  can be positive if the considered tunneling process involves the conduction band valley whose energy minimum is greater than the conduction band energy  $E_C$ .  $\Delta_C > 0$  increases the effective band gap.
- $\Delta_V$  is the valence band offset, and  $\Delta_V > 0$  increases the effective band gap.
- $\kappa$  is the magnitude of the imaginary wavevector obtained from the Kane two-band dispersion relation [33]:

$$\kappa = \frac{1}{\hbar} \sqrt{m_r E_{g,\text{tun}} (1 - \alpha^2)} \quad (500)$$

$$\alpha = -\frac{m_0}{2m_r} + 2 \sqrt{\frac{m_0}{2m_r} \frac{\epsilon - E_V + \Delta_V}{E_{g,\text{tun}}} - \frac{1}{2}} + \frac{m_0^2}{16m_r^2} + \frac{1}{4} \quad (501)$$

$$\frac{1}{m_r} = \frac{1}{m_V} + \frac{1}{m_C} \quad (502)$$

$E_{g,\text{tun}} = E_{g,\text{eff}} + \Delta_C + \Delta_V$  is the effective band gap including the band offsets, and  $k_m$  is the maximum transverse momentum determined by the maximum valence-band energy  $\epsilon_{\text{max}}$  and the minimum conduction-band energy  $\epsilon_{\text{min}}$ :

$$k_m^2 = \min(k_{vm}^2, k_{cm}^2) \quad (503)$$

$$k_{vm}^2 = \frac{2m_V(\epsilon_{\text{max}} - \epsilon)}{\hbar^2} \quad (504)$$

$$k_{\text{cm}}^2 = \frac{2m_C(\varepsilon - \varepsilon_{\text{min}})}{\hbar^2} \quad (505)$$

In the Kane two-band dispersion relation, the effective mass in the conduction band  $m_C$  and the valence band  $m_V$  are related [33]:

$$\frac{1}{m_C} = \frac{1}{2m_r} + \frac{1}{m_0} \quad (506)$$

$$\frac{1}{m_V} = \frac{1}{2m_r} - \frac{1}{m_0} \quad (507)$$

The net hole recombination rate due to the phonon-assisted band-to-band tunneling process  $R_{\text{net}}^P$  can be written as:

$$R_{\text{net}}^P = |\nabla E_V(0)| C_p \exp \left[ -2 \int_0^{x_0} \kappa_V dx - 2 \int_{x_0}^l \kappa_C dx \right] \left[ \exp \left[ \frac{\varepsilon - E_{F,n}(l)}{kT(l)} \right] + 1^{-1} - \exp \left[ \frac{\varepsilon - E_{F,p}(0)}{kT(0)} \right] + 1^{-1} \right] \quad (508)$$

$$C_p = \frac{g(1+2N_{\text{op}})D_{\text{op}}^2}{2^6 \pi^2 \rho \varepsilon_{\text{op}} E_{g,\text{tun}}} \sqrt{\frac{m_V m_C}{hl \sqrt{2m_r E_{g,\text{tun}}}}} \int_0^{x_0} \frac{dx}{\kappa_V} \int_{x_0}^l \frac{dx}{\kappa_C} \left[ 1 - \exp \left[ -k_{\text{vm}}^2 \frac{dx}{\kappa_V} \right] \right] \left[ 1 - \exp \left[ -k_{\text{cm}}^2 \frac{dx}{\kappa_C} \right] \right] \quad (509)$$

where  $D_{\text{op}}$ ,  $\varepsilon_{\text{op}}$ , and  $N_{\text{op}} = [\exp(\varepsilon_{\text{op}}/kT) - 1]^{-1}$  are the deformation potential, energy, and number of optical phonons, respectively,  $\rho$  is the mass density, and  $\kappa_V$  and  $\kappa_C$  are the magnitude of the imaginary wavevectors from the Keldysh dispersion relation:

$$\kappa_V = \frac{1}{\hbar} \sqrt{2m_V |\varepsilon - E_V + \Delta_V|} \Theta(\varepsilon - E_V + \Delta_V) \quad (510)$$

$$\kappa_C = \frac{1}{\hbar} \sqrt{2m_C |E_C + \Delta_C - \varepsilon|} \Theta(E_C + \Delta_C - \varepsilon) \quad (511)$$

and  $x_0$  is the location where  $\kappa_V = \kappa_C$ .

As Equation 499 and Equation 508 are the extension of the results in [33] to arbitrary band profiles, these expressions are reduced to the well-known Kane and Keldysh models in the uniform electric-field limit [33]:

$$R_{\text{net}} = A \frac{F_0^P}{F_0^D} \exp \left[ -\frac{B}{F} \right] \quad (512)$$

where  $F_0 = 1 \text{ V/cm}$ ,  $P = 2$  for the direct tunneling process, and  $P = 2.5$  for the phonon-assisted tunneling process.

At  $T = 300 \text{ K}$  without the bandgap narrowing effect, the prefactor  $A$  and the exponential factor  $B$  for the direct tunneling process can be expressed by [33]:

$$A = \frac{g\pi m_r^{1/2} (qF_0)^2}{9h^2 [E_g(300\text{K}) + \Delta_C + \Delta_V]^{1/2}} \quad (513)$$

## Chapter 16: Generation–Recombination

### Band-to-Band Tunneling Models

$$B = \frac{\pi^2 m_r^{1/2} [E_g(300K) + \Delta_C + \Delta_V]^{3/2}}{q h} \quad (514)$$

For the phonon-assisted tunneling process,  $A$  and  $B$  can be expressed by [33]:

$$A = \frac{g(m_V m_C)^{3/2} (1 + 2N_{op}) D_{op}^2 (qF_0)^{5/2}}{2^{21/4} h^{5/2} m_r^{5/4} \rho \epsilon_{op} [E_g(300K) + \Delta_C + \Delta_V]^{7/4}} \quad (515)$$

$$B = \frac{2^{7/2} \pi m_r^{1/2} [E_g(300K) + \Delta_C + \Delta_V]^{3/2}}{3 q h} \quad (516)$$

## Using the Nonlocal Path Band-to-Band Tunneling Model

Setting `Model=NonlocalPath` in the `Band2Band` option of the command file activates the nonlocal path band-to-band tunneling model. Multiple processes with different parameters are supported by using the `ParameterSetName` keyword and multiple named `Band2BandTunneling` parameter sets (see [Using Band-to-Band Tunneling on page 524](#)).

### Note:

Each tunneling path (characterized by a particular parameter set name) must have a consistent tunneling process (either direct or phonon-assisted) in different regions. For example, specifying `Ppath=0` (direct tunneling) in silicon regions and specifying `Ppath` greater than zero (phonon-assisted tunneling) in polysilicon regions generate an error message.

`MaxTunnelLength` in the parameter file specifies the maximum length of the tunneling path. If the length reaches `MaxTunnelLength` before a valid tunneling path is found, then band-to-band tunneling is neglected.

You can choose between two input parameter sets in the `Band2BandTunneling` section:

- The first parameter set consists of  $(m_C, m_V, g, \Delta_C, \Delta_V)$  in the case of direct tunneling or  $(m_C, m_V, gD_{op}^2/\rho, \epsilon_{op}, \Delta_C, \Delta_V)$  in the case of phonon-assisted tunneling.
- The second parameter set consists of  $(A, B, m_V/m_C, \Delta_C, \Delta_V)$  in the case of direct tunneling or  $(A, B, m_V/m_C, \epsilon_{op}, \Delta_C, \Delta_V)$  in the case of phonon-assisted tunneling.

### Note:

The temperature-specific band gap  $E_g(300K) + \Delta_C + \Delta_V$  in [Equation 513](#) to [Equation 516](#) is used solely for the definitions of the  $A$  and  $B$  parameters to be consistent with the Kane and Keldysh models in the uniform electric-field limit. Internally, the second parameter set is transformed into the first parameter set. The band gap that is used for the WKB integrals features all dependencies (for example, mole fraction, temperature, and stress) as specified in the `Physics` section.

You can specify an anisotropic electron tunneling mass according to the ellipsoidal valley band structure model. In the `Band2BandTunneling` section, the longitudinal and transversal electron tunneling masses are given by `m_c_l` and `m_c_t`, respectively. The longitudinal direction of the ellipsoid is given in crystal coordinates by `ldir_c`. For the WKB integration, Sentaurus Device computes the directional tunneling mass along the tunneling direction according to:

$$\frac{1}{m_d} = \frac{1}{\hbar^2} \frac{\partial^2 E(\vec{k})}{\partial \vec{d}^2} \quad (517)$$

Specifying  $\varepsilon_{op} = 0$  selects the direct tunneling process. When  $\varepsilon_{op} = 0$  and the option `FranzDispersion` is switched on in the `Band2Band` option of the command file, the magnitude of the imaginary wavevector is obtained from the Franz two-band dispersion relation ([Equation 823](#)) instead of the Kane two-band dispersion relation ([Equation 500](#)).

Specifying  $\varepsilon_{op} > 0$  selects the phonon-assisted tunneling process. When  $m_V/m_C = 0$ ,  $m_V$  and  $m_C$  are determined from [Equation 506](#) and [Equation 507](#).

In the parameter file, the parameter pairs `QuantumPotentialFactor` and `QuantumPotentialPosFac` in the `Band2BandTunneling` parameter set specify the prefactors for the electron and hole quantum potentials obtained from the density gradient model (see [Density Gradient Model on page 362](#)) that can be added to the effective band gap for tunneling. By default, they are zero, such that the quantum potentials are neglected in the computation of the generation rate. When they are nonzero, the quantum potentials multiplied by the corresponding prefactors are added to the conduction and valence band edges when the transmission coefficient is computed. For `QuantumPotentialFactor`, an addition is performed irrespective of the sign of the quantum potential. For `QuantumPotentialPosFac`, an addition is performed only when the quantum potential is positive, that is, only where quantization causes an effective widening of the band gap. For example, the following section causes the electron and hole quantum potentials to be added to the band gap wherever they cause an effective widening of the band gap:

```
Band2BandTunneling {
    ...
    QuantumPotentialPosFac = 1 1
}
```

[Table 97](#) lists the model parameters, which can be mole fraction dependent. The default values for parameters  $A$  and  $B$  were obtained from [\[31\]](#).

*Table 97 Parameters of nonlocal path band-to-band tunneling model*

Symbol	Parameter name	Default value	Unit
$A$	<code>Apath</code>	$4 \times 10^{14}$	$\text{cm}^{-3}\text{s}^{-1}$
$B$	<code>Bpath</code>	$1.9 \times 10^7$	$\text{V}\text{cm}^{-1}$

*Table 97 Parameters of nonlocal path band-to-band tunneling model (Continued)*

Symbol	Parameter name	Default value	Unit
$gD_{\text{op}}^2/\rho$	Cpath	0	$\text{J}^2 \text{cmkg}^{-1}$
$g$	degeneracy	0	1
$\Delta_C$	Dcpath	0	eV
$\Delta_V$	Dvpath	0	eV
$\vec{l}_{\text{dir},c}$	ldir_c	(0,0,0)	1
$m_C$	m_c	0	$m_0$
$m_{C,l}$	m_c_l	0	$m_0$
$m_{C,t}$	m_c_t	0	$m_0$
$m_V$	m_v	0	$m_0$
$m_{C,q}$	m_c_q	0.26	$m_0$
$m_{V,q}$	m_v_q	0.16	$m_0$
$\epsilon_{\text{op}}$	Ppath	0.037	eV
$m_V/m_C$	Rpath	0	1

## Handling Derivatives

Due to the dynamic search for tunnel paths, the structure of the system Jacobian depends on solution variables. Therefore, the nonzero entries of the Jacobian are not known *a priori*. The possible entries are estimated before entering the Newton loop.

The computation of nonlocal derivative terms is controlled in the global `Math` section, in the Quasistationary statement (see [Quasistationary Ramps on page 122](#)) and in the Transient statement (see [Transient Ramps on page 144](#)) using the `NonlocalPath` section.

Using the `Derivative` option in the `NonlocalPath` section, the computation of nonlocal derivatives can be switched on (`Derivative=1`) and switched off (`Derivative=0`). By default, the computation of nonlocal derivatives is switched off.

If nonlocal derivatives are computed, then different strategies are available, using the `Strategy` option, to fill the system Jacobian:

- If `Strategy=1`, then all previously collected Jacobian entries are registered as possible nonzero entries for the next Newton loop. The collection bin of possible Jacobian entries is emptied at the beginning of the `Quasistationary` and `Transient` ramps.
- If `Strategy=2`, then Jacobian entries that occur only during the last `N` Newton iterations are registered as possible nonzero entries for the next Newton loop.
- If `Strategy=3`, then it automatically activates and deactivates the computation of nonlocal derivatives during the `Quasistationary` and `Transient` ramps, depending on the step size. If the step size is smaller than `MinStep`, nonlocal derivatives are computed. If the step size is greater than `MaxStep`, nonlocal derivatives are not computed. Jacobian entries that occur during the last `N` Newton iterations are registered as possible nonzero entries for the next Newton loop.

For example:

```
NonlocalPath (
    Derivative=1
    Strategy=3
    N=5
    MinStep=1.0e-5
    MaxStep=1.0e-2
)
```

**Note:**

Including nonlocal derivatives is computationally heavy and reduces the sparsity of the system Jacobian. Simulation times will increase.

## Tunneling Parameters From the Physical Model Interface

You can create a tunneling parameters PMI model for the customized computation of `DcPath`, `DvPath`, `m_c`, and `m_v` (see [Tunneling Parameters on page 1276](#)). When the computation of nonlocal derivatives is switched on (`Derivative=1`), nonlocal derivatives with respect to the lattice temperature are included automatically.

## Energy Shift due to Geometric Confinement

If carriers are confined in thin semiconductor layers, then geometric quantization leads to an energy shift of the conduction and valence bands leading to a larger band gap. The quantum-mechanical energy shift is described by the “particle in a box” model, which describes a particle in a one-dimensional infinite potential well.

## Chapter 16: Generation–Recombination

### Band-to-Band Tunneling Models

For a parabolic dispersion, the energy shift can be described as:

$$\Delta E_q = \frac{\hbar^2}{2m_q} \frac{\pi^2}{L^2} \cdot \quad (518)$$

where:

- $L$  is the box length and is computed with the `LayerThickness` command (see [Extracting Layer Thickness on page 379](#)).
- $m_q$  is the quantization mass.

This energy shift due to geometric confinement can be switched on (`LayerThickness`) and switched off (`-LayerThickness`) in the `NonlocalPath` section.

#### Note:

You must also switch on the `LayerThickness` command in the corresponding `Physics` section.

You can specify the electron (`m_c_q`) and hole (`m_v_q`) quantization masses in the `Band2BandTunneling` section in the parameter file.

## Postprocessing Mode

To estimate the nonlocal tunneling generation rate without convergence difficulties and without computationally heavy nonlocal derivatives, Sentaurus Device provides the postprocessing mode of the dynamic nonlocal path band-to-band tunneling model.

The postprocessing mode can be switched on (`Postprocessing`) and switched off (`-Postprocessing`) in the `NonlocalPath` section. By default, the postprocessing mode is switched off. Specifying the `NonlocalPath` section in the `Quasistationary` or `Transient` statement overwrites the specifications of the `NonlocalPath` section in the global `Math` section.

In the postprocessing mode, generation rates due to the dynamic nonlocal path band-to-band tunneling model are excluded from the continuity equations. Based on this solution without nonlocal tunneling, Sentaurus Device calculates the nonlocal tunneling generation rates in a postprocessing step.

#### Note:

In the postprocessing mode, nonlocal tunneling is not computed self-consistently.

There is no feedback of the nonlocal tunneling generation rates on the solution that has been used to calculate the nonlocal tunneling generation rates.

The tunneling generation currents are computed according to  $J^{n/p} = q R_{\text{net}}^{n/p}(r) d^3 r$  where the integration volume expands over all semiconductor regions.

## Chapter 16: Generation–Recombination

### Bimolecular Recombination Model

If postprocessing is activated, then the electron and hole tunneling generation currents `eBand2BandGenerationCurrent` and `hBand2BandGenerationCurrent` are written in units of C/s into the plot file.

## Frozen Tunneling Direction

In some cases, the estimation of the tunneling direction pointing along the negative gradient of the valence band energy can be a tunneling direction of minor importance. Furthermore, freezing the tunneling direction can support numeric stability to achieve convergence. You can specify the tunneling direction by setting `direction=<vector>` in the `NonlocalPath` section.

## Visualizing Nonlocal Band-to-Band Generation Rate

To plot the electron and hole generation rates, specify `eBand2BandGeneration` and `hBand2BandGeneration` in the `Plot` section, respectively.

### Note:

`Band2BandGeneration` is equal to `hBand2BandGeneration`.

---

## Bimolecular Recombination Model

The bimolecular recombination model describes the interaction of electron–hole pairs and singlet excitons (see [Singlet Exciton Equation on page 291](#)).

The rate of electron–hole pair and singlet exciton recombination follows the Langevin form, that is, it is proportional to the carrier mobility. The bimolecular recombination rate is given by:

$$R_{\text{bimolec}} = \gamma \cdot \frac{q}{\epsilon_0 \epsilon_r} \cdot (\mu_n + \mu_p) \cdot np - n_{i,\text{eff}}^2 \frac{n_{se}}{n_{se}^{\text{eq}}} \quad (519)$$

where:

- $\gamma$  is a prefactor for the singlet exciton.
- $q$  is the elementary charge.
- $\epsilon_0$  and  $\epsilon_r$  denote the free space and relative permittivities, respectively.
- Electron and hole mobilities are given by  $\mu_n$  and  $\mu_p$ , accordingly.
- $n$ ,  $p$ , and  $n_{i,\text{eff}}$  describe the electron, hole, and effective intrinsic density, respectively.
- $n_{se}$  is the singlet exciton density.
- $n_{se}^{\text{eq}}$  denotes the singlet-exciton equilibrium density.

## Using the Bimolecular Recombination Model

You activate the model by specifying `Bimolecular` as an argument of the `Recombination` statement in the `SingletExciton` section (see [Table 260 on page 1650](#)). It is switched off by default and can be activated regionwise (see [Singlet Exciton Equation on page 291](#)). For example:

```
Physics (Region="EML-ETL") {
    SingletExciton (
        Recombination ( Bimolecular )
    )
}
```

The model parameter is accessible in the `SingletExciton` section of the parameter file.

*Table 98 Parameter of bimolecular recombination model*

Symbol	Parameter name	Default value	Unit
$\gamma$	gamma	0.25	1

## Exciton Dissociation Model

The exciton dissociation model describes the dissociation of singlet excitons into electron–hole pairs at semiconductor–semiconductor and semiconductor–insulator interfaces.

The rate of singlet exciton interface dissociation is modeled as:

$$R_{\text{se,diss}}^{\text{surf}} = v_{\text{se},0} \sigma_{\text{se}-N_{\text{diss}}} N_{\text{se,diss}}^{\text{surf}} (n_{\text{se}} - n_{\text{se}}^{\text{eq}}) \quad (520)$$

where:

- $v_{\text{diss}}^{\text{surf}} = v_{\text{se},0} \sigma_{\text{se}-N_{\text{diss}}} N_{\text{se,diss}}^{\text{surf}}$  is the singlet exciton recombination velocity in cm/s.
- $\sigma_{\text{se}-N_{\text{diss}}}^{\text{surf}}$  is the capture cross-section of exciton dissociation centers with the surface density  $N_{\text{se,diss}}^{\text{surf}}$ .
- $v_{\text{se},0}$  is the singlet exciton thermal velocity.
- $n_{\text{se}}$  and  $n_{\text{se}}^{\text{eq}}$  are the exciton and equilibrium exciton densities, respectively.

In the dissociation process, a singlet exciton generates an electron–hole pair. The rate in [Equation 520](#) is a recombination rate for the singlet exciton equation and a generation rate for the electron and hole continuity equations.

---

## Using the Exciton Dissociation Model

The model is activated using the keyword `Dissociation` as an argument of the `Recombination` statement in the `SingletExciton` section (see [Table 260 on page 1650](#)). It is switched off by default and can be activated regionwise (see [Singlet Exciton Equation on page 291](#)). The following example activates exciton dissociation at the `EML/ETL` region interface:

```
Physics (RegionInterface="EML/ETL") {
    SingletExciton (Recombination ( Dissociation ))
}
```

The model parameters are accessible in the `SingletExciton` section of the parameter file.

*Table 99 Parameters of exciton dissociation model*

Symbol	Parameter name	Default value	Unit
$\sigma_{se - N_{diss}}$	<code>ex_dXsection</code>	$1 \times 10^{-8}$	$\text{cm}^2$
$N_{se,diss}^{\text{surf}}$	<code>N_diss</code>	$1 \times 10^{10}$	$\text{cm}^{-2}$

---

## References

- [1] D. J. Roulston, N. D. Arora, and S. G. Chamberlain, “Modeling and Measurement of Minority-Carrier Lifetime versus Doping in Diffused Layers of n+p Silicon Diodes,” *IEEE Transactions on Electron Devices*, vol. ED-29, no. 2, pp. 284–291, 1982.
- [2] J. G. Fossum, “Computer-Aided Numerical Analysis of Silicon Solar Cells,” *Solid-State Electronics*, vol. 19, no. 4, pp. 269–277, 1976.
- [3] J. G. Fossum and D. S. Lee, “A Physical Model for the Dependence of Carrier Lifetime on Doping Density in Nondegenerate Silicon,” *Solid-State Electronics*, vol. 25, no. 8, pp. 741–747, 1982.
- [4] J. G. Fossum *et al.*, “Carrier Recombination and Lifetime in Highly Doped Silicon,” *Solid-State Electronics*, vol. 26, no. 6, pp. 569–576, 1983.
- [5] A. Nakagawa, “One-dimensional device model of the npn bipolar transistor including heavy doping effects under Fermi statistics,” *Solid-State Electronics*, vol. 22, no. 11, pp. 943–949, 1979.
- [6] M. S. Tyagi and R. Van Overstraeten, “Minority Carrier Recombination in Heavily-Doped Silicon,” *Solid-State Electronics*, vol. 26, no. 6, pp. 577–597, 1983.

## Chapter 16: Generation–Recombination

### References

- [7] H. Goebel and K. Hoffmann, "Full Dynamic Power Diode Model Including Temperature Behavior for Use in Circuit Simulators," in *Proceedings of the 4th International Symposium on Power Semiconductor Devices & ICs (ISPSD)*, Tokyo, Japan, pp. 130–135, May 1992.
- [8] A. Schenk, "A Model for the Field and Temperature Dependence of Shockley–Read–Hall Lifetimes in Silicon," *Solid-State Electronics*, vol. 35, no. 11, pp. 1585–1596, 1992.
- [9] R. R. King, R. A. Sinton, and R. M. Swanson, "Studies of Diffused Phosphorus Emitters: Saturation Current, Surface Recombination Velocity, and Quantum Efficiency," *IEEE Transactions on Electron Devices*, vol. 37, no. 2, pp. 365–371, 1990.
- [10] R. R. King and R. M. Swanson, "Studies of Diffused Boron Emitters: Saturation Current, Bandgap Narrowing, and Surface Recombination Velocity," *IEEE Transactions on Electron Devices*, vol. 38, no. 6, pp. 1399–1409, 1991.
- [11] A. Cuevas *et al.*, "Surface Recombination Velocity and Energy Bandgap Narrowing of Highly Doped n-Type Silicon," in *13th European Photovoltaic Solar Energy Conference*, Nice, France, pp. 337–342, October 1995.
- [12] A. Schenk and U. Krumbein, "Coupled defect-level recombination: Theory and application to anomalous diode characteristics," *Journal of Applied Physics*, vol. 78, no. 5, pp. 3185–3192, 1995.
- [13] L. Huldt, N. G. Nilsson, and K. G. Svantesson, "The temperature dependence of band-to-band Auger recombination in silicon," *Applied Physics Letters*, vol. 35, no. 10, pp. 776–777, 1979.
- [14] W. Lochmann and A. Haug, "Phonon-Assisted Auger Recombination in Si with Direct Calculation of the Overlap Integrals," *Solid State Communications*, vol. 35, no. 7, pp. 553–556, 1980.
- [15] R. Häcker and A. Hangleiter, "Intrinsic upper limits of the carrier lifetime in silicon," *Journal of Applied Physics*, vol. 75, no. 11, pp. 7570–7572, 1994.
- [16] A. Richter *et al.*, "Improved parameterization of Auger recombination in silicon," *Energy Procedia*, vol. 27, pp. 88–94, April 2012.
- [17] P. P. Altermatt *et al.*, "Injection dependence of spontaneous radiative recombination in c-Si: experiment, theoretical analysis, and simulation," in *Proceedings of the 5th International Conference on Numerical Simulation of Optoelectronic Devices (NUSOD)*, Berlin, Germany, pp. 47–48, September 2005.
- [18] A. G. Chynoweth, "Ionization Rates for Electrons and Holes in Silicon," *Physical Review*, vol. 109, no. 5, pp. 1537–1540, 1958.
- [19] R. van Overstraeten and H. de Man, "Measurement of the Ionization Rates in Diffused Silicon p-n Junctions," *Solid-State Electronics*, vol. 13, no. 1, pp. 583–608, 1970.

## Chapter 16: Generation–Recombination

### References

- [20] Y. Okuto and C. R. Crowell, "Threshold Energy Effect on Avalanche Breakdown Voltage in Semiconductor Junctions," *Solid-State Electronics*, vol. 18, no. 2, pp. 161–168, 1975.
- [21] T. Lackner, "Avalanche Multiplication in Semiconductors: A Modification of Chynoweth's Law," *Solid-State Electronics*, vol. 34, no. 1, pp. 33–42, 1991.
- [22] M. C. Vecchi and M. Rudan, "Modeling Electron and Hole Transport with Full-Band Structure Effects by Means of the Spherical-Harmonics Expansion of the BTE," *IEEE Transactions on Electron Devices*, vol. 45, no. 1, pp. 230–238, 1998.
- [23] S. Reggiani *et al.*, "Electron and Hole Mobility in Silicon at Large Operating Temperatures—Part I: Bulk Mobility," *IEEE Transactions on Electron Devices*, vol. 49, no. 3, pp. 490–499, 2002.
- [24] M. Valdinoci *et al.*, "Impact-ionization in silicon at large operating temperature," in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, Kyoto, Japan, pp. 27–30, September 1999.
- [25] E. Gnani *et al.*, "Extraction method for the impact-ionization multiplication factor in silicon at large operating temperatures," in *Proceedings of the 32nd European Solid-State Device Research Conference (ESSDERC)*, Florence, Italy, pp. 227–230, September 2002.
- [26] S. Reggiani *et al.*, "Investigation about the High-Temperature Impact-Ionization Coefficient in Silicon," in *Proceedings of the 34th European Solid-State Device Research Conference (ESSDERC)*, Leuven, Belgium, pp. 245–248, September 2004.
- [27] S. Reggiani *et al.*, "Experimental extraction of the electron impact-ionization coefficient at large operating temperatures," in *IEDM Technical Digest*, San Francisco, CA, USA, pp. 407–410, December 2004.
- [28] T. Hatakeyama *et al.*, "Physical Modeling and Scaling Properties of 4H-SiC Power Devices," in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, Tokyo, Japan, pp. 171–174, September 2005.
- [29] R. J. McIntyre, "On the Avalanche Initiation Probability of Avalanche Diodes Above the Breakdown Voltage," *IEEE Transactions on Electron Devices*, vol. ED-20, no. 7, pp. 637–641, 1973.
- [30] A. Schenk, "Rigorous Theory and Simplified Model of the Band-to-Band Tunneling in Silicon," *Solid-State Electronics*, vol. 36, no. 1, pp. 19–34, 1993.
- [31] G. A. M. Hurkx, D. B. M. Klaassen, and M. P. G. Knuvers, "A New Recombination Model for Device Simulation Including Tunneling," *IEEE Transactions on Electron Devices*, vol. 39, no. 2, pp. 331–338, 1992.
- [32] J. J. Liou, "Modeling the Tunnelling Current in Reverse-Biased p/n Junctions," *Solid-State Electronics*, vol. 33, no. 7, pp. 971–972, 1990.
- [33] E. O. Kane, "Theory of Tunneling," *Journal of Applied Physics*, vol. 32, no. 1, pp. 83–91, 1961.

## Traps and Fixed Charges

---

*This chapter describes how traps are handled by Sentaurus Device.*

---

### Introduction to Traps

Traps are important in device physics because they provide doping, enhance recombination, and increase leakage through insulators. Several models such as Shockley–Read–Hall recombination depend on traps implicitly, but do not actually model them. This chapter describes models that take the occupation and the space charge stored on traps explicitly into account. It also describes the specification of fixed charges.

Sentaurus Device handles several trap types and different types of energetic distribution, and provides various models for capture and emission rates. Traps are available for both bulk regions and interfaces. There is also a simple model restricted to insulator fixed charges (see [Insulator Fixed Charges on page 582](#)).

---

### Syntax for Traps

You specify trap distributions and trap models in the `Physics` section of the command file. In contrast to other models, most trap model parameters are also specified in this section. Parameter specifications in the parameter file serve as defaults for those in the command file.

Traps can be specified for interfaces or bulk regions. Specifications take the form:

```
Physics (Region="gobbledygook") {
    Traps(
        ( <trap_specification> )
        ( <trap_specification> )
        ...
    )
}
```

The same form applies to `Material`, `RegionInterface`, and `MaterialInterface`.

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

Each <trap\_specification> describes one particular trap distribution and the models and parameters applied to it. [Table 331 on page 1701](#) summarizes the options that can appear in trap specifications.

When only one trap specification is present, you can omit the inner parentheses. For example:

```
Physics (Region="gobbledygook") {
    Traps(
        <trap_specification>
    )
}
```

#### Note:

Wherever a contact exists at a specified region interface, Sentaurus Device does not recognize the interface traps within the bounds of the contact because the contact itself constitutes a region and effectively overwrites the interface between the two *material* regions. This is true even if the contact is not declared in the Electrode section.

---

## Trap Types

The following keywords select the trap type:

- `FixedCharge` traps are always completely occupied.
- `Acceptor` and `eNeutral` traps are uncharged when unoccupied, and they carry the charge of one electron when fully occupied.
- `Donor` and `hNeutral` traps are uncharged when unoccupied, and they carry the charge of one hole when fully occupied.

---

## Energetic and Spatial Distribution of Traps

The following keywords determine the energetic distribution of traps:

- `Level` selects a single-energy level.
- `Uniform` selects a uniform distribution.
- `Exponential` selects an exponential distribution.
- `Gaussian` selects a Gaussian distribution.
- `Table` selects a user-defined table distribution.
- `RandomizeEnergy` selects a random single energy level from the `Uniform`, `Gaussian`, or `Table` distributions.

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

The relevant equations for the different trap distributions are:

$$\begin{aligned}
 N_0 & \quad \text{for } E = E_0 && \text{for Level} \\
 N_0 & \quad \text{for } E_0 - 0.5E_S < E < E_0 + 0.5E_S && \text{for Uniform} \\
 N_0 \exp \left| \frac{E - E_0}{E_S} \right| & && \text{for Exponential} \\
 N_0 \exp \frac{(E - E_0)^2}{2E_S^2} & && \text{for Gaussian} \\
 \\ 
 N_1 & \quad \text{for } E = E_1 && \\
 \dots & \quad \dots && \text{for Table} \\
 N_m & \quad \text{for } E = E_m &&
 \end{aligned} \tag{521}$$

$N_0$  is set with the `Conc` keyword as follows:

- For a `Level` distribution, `Conc` is given in  $\text{cm}^{-3}$  (for regionwise or materialwise specifications) or  $\text{cm}^{-2}$  (for interface-wise specifications). For other distributions, `Conc` is given in  $\text{eV}^{-1} \text{cm}^{-3}$  or  $\text{eV}^{-1} \text{cm}^{-2}$ .
- For `FixedCharge` traps, the sign of `Conc` denotes the sign of the fixed charges. For other trap types, `Conc` must not be negative.
- For `DiscreteTrapT2T`, `Conc` has a different meaning and is used to scale the defined discrete traps (see [Trap-to-Trap Tunneling \(DiscreteTrapT2T\) on page 561](#)).

For a `Table` distribution, individual levels are given as a pair of energies (in eV) and the corresponding concentrations (in  $\text{eV}^{-1} \text{cm}^{-3}$  or  $\text{eV}^{-1} \text{cm}^{-2}$ ). Depending on the presence of the `fromCondBand`, `fromMidBandGap`, or `fromValBand` keyword, energy levels in the table are relative to the conduction band, intrinsic energy, or valence band, respectively. To obtain the absolute concentration (in  $\text{cm}^{-3}$  or  $\text{cm}^{-2}$ ) for each level, the energy range between the smallest and the largest energy in the table is split into intervals. The boundaries of the intervals are the energies that are in the middle between adjacent energies in the table. The absolute concentration of a level is the product of the size of the energy interval that contains it and the concentration specified in the table. If the table contains only one level, an interval size of 1 eV is assumed.

The following example creates a trap level at the midgap with a concentration of  $0.1 \text{ eV} \times 10^{16} \text{ cm}^{-3} \text{ eV}^{-1} = 10^{15} \text{ cm}^{-3}$ , and a level at both 0.1 eV above and below the midgap, with a concentration of  $0.05 \text{ eV} \times 10^{15} \text{ cm}^{-3} \text{ eV}^{-1} = 5 \times 10^{13} \text{ cm}^{-3}$ :

```
Traps ((Table=(-0.1 1e15 0 1e16 0.1 1e15) fromMidBandGap))
```

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

In [Equation 521](#),  $E_0$  and  $E_S$  are given in eV by `EnergyMid` and `EnergySig`. The energy of the center of the trap distribution,  $E_{\text{trap}}^0$ , is obtained from  $E_0$  depending on the presence of one of the following keywords:

$$E_{\text{trap}}^0 = \begin{cases} E_C - E_0 - E_{\text{shift}} & \text{fromCondBand} \\ [E_C + E_V + kT \ln(N_V/N_C)]/2 + E_0 + E_{\text{shift}} & \text{fromMidBandGap} \\ E_V + E_0 + E_{\text{shift}} & \text{fromValBand} \end{cases} \quad (522)$$

Internally, Sentaurus Device approximates trap energy distributions by discrete energy levels. The default number of levels is 13 and is set by `TrapDLN` in the `Math` section. A peak sampling method is available for the Gaussian and Exponential distributions, which can be activated with the syntax `TrapDLN(PeakSampling)=<int>` in the `Math` section.

#### Note:

You are encouraged to use `TrapDLN(PeakSampling)=<int>` as this method provides a better discretization of the Gaussian and exponential trap energy distributions.

In [Equation 522](#),  $E_{\text{shift}}$  is zero (default) or is computed by a physical model interface (PMI) specified in one of the following ways:

```
EnergyShift=<model_name>  
EnergyShift=(<model_name>, <int>)
```

The PMI depends on the electric field and the lattice temperature. By default, these quantities are taken at the location of the trap. Alternatively, you can use `ReferencePoint=<vector>` to specify a coordinate in the device from where these quantities should be taken. See [Trap Energy Shift on page 1396](#).

In a region for continuous traps with a defined trap energy distribution, every vertex of the trap region inherits that energy distribution. Alternatively, you can use a random trap energy distribution (`RandomizeEnergy`) whereby, at each vertex, there is only one trap energy level (chosen randomly from the energy distribution). The supported energy distributions are Uniform, Gaussian, and Table. The sampling is performed such that, when you collect the trap energy level defined in all vertices, there will be more samples located at the peak of the energy distribution. In the case of a `Table` definition for `RandomizeEnergy`, the first column of the table denotes the trap energy level and the second column is the associated probability of randomly choosing that energy level. Therefore, the sum of all the probability entries in the second column must add up to one. Note that this differs from the usual `Table` energy distribution, whereby the second column denotes the concentration.

For traps located at interfaces, `Region` or `Material` allows you to specify the region or material on one side of the interface; the energy specification then refers to the band structure on that side. For heterointerfaces, the charge and recombination rates due to traps are fully accounted for on that side. Without this side specification, the energy parameters

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

refer to the bands that you see when you plot the band edges, which usually originate from the lower bandgap material. The charge and recombination rates due to traps are evenly distributed on both sides.

Note that if the trap energies are not specified relative to the band edges of the material with the lower bandgap, the correction of trap energies is performed at 300K. This implies that if you perform a temperature-dependent simulation with traps at an interface between materials that have different temperature dependencies, you should use both heterointerfaces and Region or Material for the side specification, to make the correct specification of trap energies easier.

By default, trap energies refer to effective band edges, that is, band edges that include bandgap narrowing. By specifying Reference=BandGap, trap energies refer to band edges that exclude bandgap narrowing.

The Cutoff keyword controls truncation and applies only to traps with Uniform, Gaussian, and Exponential distributions:

- Cutoff=BandGap truncates to the band gap at 300 K without bandgap narrowing.
- Cutoff=EffectiveBandGap (default) truncates the trap energy distribution to the effective bandgap at 300 K.
- Cutoff=None deactivates truncation.
- Cutoff=Simple activates a truncation approach used by previous versions of Sentaurus Device, which ignores mole fraction dependency of the band gap.

#### Note:

Truncation does not affect the band edges to which trap energies refer during simulation. These are always the solution-dependent band edges including temperature dependency. Traps with Level and Table distributions are never truncated to the band gap.

By default, trap concentrations are uniform over the domain for which they are specified. With SFactor= "<dataset\_name>", the given dataset determines the spatial distribution. If Conc is zero or is omitted, the dataset determines the density directly. For Uniform, Gaussian, and Exponential trap distributions, this is the total density (in  $\text{cm}^{-3}$  or  $\text{cm}^{-2}$ ) of the integrated energy distribution. If Conc is nonzero, the SFactor is scaled by the maximum of its absolute value and multiplied by  $N_0$ . Datasets available for SFactor are:

- DeepLevels, xMoleFraction, and yMoleFraction (read from the doping file)
- TotalTrapConcentration, eTrappedCharge, and hTrappedCharge (read from the file specified by DevFields in the File section)
- PMI user fields (read from the file specified by PMIUserFields in the File section)

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

If a `Physics` section specifies a location on `MaterialInterface` or `RegionInterface`, and the dataset for `SFactor` is `DeepLevels`, `eTrappedCharge`, `hTrappedCharge`, or `TotalTrapConcentration`, the input TDR file must have the corresponding fields on the interfaces (see [Interface Plots on page 180](#)).

Alternatively, with `SFactor= "<pmi_model_name>"`, the spatial distribution is computed using the PMI (see [Space Factor on page 1479](#)). During a transient simulation, this PMI allows you to have a time-dependent trap concentration. In this case, the spatial distribution can be computed based on the solution from the previous time step available through the PMI.

The `SpatialShape` keyword selects a multiplicative modifier function for the trap concentration. If `SpatialShape=Uniform` (default), then the multiplier for point  $(x, y, z)$  is:

$$\Theta(\sigma_x - |x - x_0|) \Theta(\sigma_y - |y - y_0|) \Theta(\sigma_z - |z - z_0|) \quad (523)$$

If `SpatialShape=Gaussian`, then the multiplier is:

$$\exp -\frac{(x - x_0)^2}{2\sigma_x^2} - \frac{(y - y_0)^2}{2\sigma_y^2} - \frac{(z - z_0)^2}{2\sigma_z^2} \quad (524)$$

In [Equation 523](#) and [Equation 524](#),  $(x_0, y_0, z_0)$  and  $(\sigma_x, \sigma_y, \sigma_z)$  are given (in  $\mu\text{m}$ ) by the `SpaceMid` and `SpaceSig` keywords, respectively. By default, the components of `SpaceSig` are huge, so that the multiplier becomes one.

---

## Specifying Single Traps

Including `SingleTrap` in a trap specification allows you to simplify the specification of parameters to mimic the behavior of a single trap:

- `SingleTrap` is allowed only with `Level` distributions for traps.
- The coordinates specified by `SpaceMid=(x0, y0, z0)` snap to the nearest vertex in the material, region, or interface that is specified as part of the `Physics` section. If a global `Physics` specification is used, the coordinates snap to the nearest vertex in the device.
- `SpatialShape=Uniform` is used automatically for the trap. You do not need to specify it.
- `SpaceSig=(1e-6, 1e-6, 1e-6)` is used automatically for the trap. This is intended to confine the trap to a single vertex. You do not need to specify it.
- The trap concentration is computed automatically such that a filled trap corresponds to one electronic charge. If you specify `Conc`, it is ignored.
- When `SingleTrap` is specified for a `FixedCharge` trap, the sign of `Conc` (if specified) is used for the sign of the fixed charge. If `Conc` is omitted or `Conc=0` is specified, the fixed charge is positive. To obtain a negative fixed charge with `SingleTrap`, specify any negative value for `Conc`.

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

- For 2D simulations, `AreaFactor` specified in the `Physics` section is used for the z-direction width when calculating the concentration.
- If specified with the random telegraph noise (RTN) option `SingleTrap(RTN)`, then access and set functions are available for mesh-based PMIs (see [Runtime Support for Mesh-Based PMI Models on page 1245](#)).

### Example

Place two single eNeutral traps at the silicon–oxide interface:

```
Physics (MaterialInterface="Silicon/Oxide") {
    Traps (
        (SingleTrap eNeutral Level EnergyMid=0 fromMidBandGap
         SpaceMid=(0.0,0.0,0.1))
        (SingleTrap eNeutral Level EnergyMid=0 fromMidBandGap
         SpaceMid=(0.2,0.0,0.3))
    )
}
```

---

## Trap Randomization

You can randomize the spatial distribution of traps by specifying `Randomize` in the trap specification, with or without an integer value:

- `Randomize` is allowed only with `Level` distributions for traps or `DiscreteTrapT2T`.
- If you specify `SingleTrap`, the location of the single trap is randomized in the material, region, or interface that is specified as part of the `Physics` section. If a global `Physics` specification is used, then the location of this trap is randomized in the entire device. If `SpaceMid` is specified, then it is ignored.
- If you do not specify `SingleTrap`, then the concentration of traps at each vertex is randomized. This is accomplished by first determining the average number of traps at a vertex based on the spatial distribution created from the trap specification. Then, this is used as the expectation value for a Poisson-distribution random number generator to obtain a new random number of traps at the vertex. Finally, this is converted back to a trap concentration.
- If you specify `DiscreteTrapT2T` for trap-to-trap tunneling, then you need an additional keyword `NumberOfRandomTraps=<int>` to indicate the number of traps to be randomly distributed in the region (see [Trap-to-Trap Tunneling \(DiscreteTrapT2T\) on page 561](#)).
- If you specify `Randomize`, then the randomization differs every time you execute the command file. You can change this behavior, however, by specifying `Randomize` with an integer value:
  - `Randomize < 0`: No randomization occurs.

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

- Randomize = 0: Same as Randomize without a value. The randomization differs for every execution of the command file.
- Randomize > 0: The specified number is used as the seed for the random number generator. This allows a particular randomization to be repeated for repeated executions of the same command file on the same platform.

### Examples

Randomize the location of one electron trap at the silicon–oxide interface:

```
Physics (MaterialInterface="Silicon/Oxide") {  
    Traps (  
        SingleTrap Randomize eNeutral Level EnergyMid=0 fromMidBandGap  
    )  
}
```

Randomize a Gaussian spatial distribution of traps in silicon:

```
Physics (Material="Silicon") {  
    Traps (  
        eNeutral Conc=1e16 Level EnergyMid=0 fromMidBandGap Randomize  
        SpatialShape=Gaussian SpaceMid=(0.0,0.0,0.05)  
        SpaceSig=(0.05,0.05,0.05)  
    )  
}
```

---

## Explicit Trap Occupation

When investigating time-delay effects, for example, it might be advantageous to start a transient simulation from an initial state with either totally empty or totally filled trap states. Depending on the position of the equilibrium Fermi level, however, it might be impossible to reach such an initial state from steady-state or quasistationary simulations (for example, it might be a metastable state with a very long lifetime).

Different mechanisms are available to initialize trap occupancies in the `Solve` section.

#### Note:

An earlier mechanism is invoked by `TrapFilling` in the `Set` and `Unset` statements of the `Solve` section. [Table 354 on page 1735](#) summarizes the syntax of these statements. This mechanism is deprecated with the exception of the `-Degradation` option.

The mechanism invoked by `Traps` in the `Set` statement in a `Solve` section allows you to set trap occupancies for individually named traps to spatial- and solution-independent values, and to freeze and unfreeze the trap occupancies of all traps. For descriptions of the options of `Traps` in the `Set` statement, see [Table 354](#).

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

To set a trap, you must define an identifying `Name` in its definition in `Traps`, which enables you to reference this trap in the `Set` command. The `Frozen` option freezes the trap occupancies for subsequent `Solve` sections, and the `-Frozen` option unfreezes them. Freezing traps implies that the traps are decoupled from both the conduction band and valence band, that is, their recombination terms are set to zero. Note that `Frozen` applies to all defined traps. Traps not explicitly initialized in the `Set` command are frozen at their actual values. For example:

```
Device "MOS" {
    Physics { ...
        Traps ( (Name="t1" eNeutral ...) (Name="t2" hNeutral ...) ... )
    }
}
System { ...
    MOS "mos1" ( ... ) { ... }
}
Solve { ...
    Set ( Traps ( "mos1"."t1" = 1. Frozen ) )
    ...
    Set ( Traps ( -Frozen ) )
    ...
}
```

In this example, some traps are named. In the first `Set` statement, trap "`t1`" of device "`mos1`" is explicitly set to one. Due to the `Frozen` option, trap "`t1`" is frozen at the specified value and trap "`t2`" is frozen at its actual value. The second `Set` statement releases all traps.

To set all the trap occupancies to a certain value between 0 and 1, you must specify the `Value` keyword as an option for `Traps`. In this case, the occupancy of all the traps (with or without names) is set to the value specified by the `Value` keyword.

In the following examples, trap occupancies are set to 1 and they are released later in the simulation using the `-Frozen` option:

```
Solve { ...
    Set ( Traps ( "mos1".value = 1. Frozen ) )
    ...
    Set ( Traps ( -Frozen ) )
    ...
}
```

or:

```
Solve { ...
    Set ( Traps ( value = 1. Frozen ) )
    ...
    Set ( Traps ( -Frozen ) )
    ...
}
```

## Chapter 17: Traps and Fixed Charges

### Introduction to Traps

#### Note:

Freezing and unfreezing of traps using the `Solve-Set-Traps` command implies freezing and unfreezing of multistate configurations, respectively (see [Manipulating MSCs During Solve on page 597](#)). If the incomplete ionization model is used, dopant ionization is affected as well (see [Chapter 13 on page 339](#)).

---

## Options to Include Traps in Doping

The effect of traps on some physical models can be treated in an approximate manner by treating trap concentration as additional doping. The following options are available for this purpose:

```
Traps (
    <trap_specification>
    [Add2TotalDoping | Add2TotalDoping(ChargedTraps)]
)
```

If you specify `Add2TotalDoping`, then the trap concentration is added to both the acceptor (or donor) doping concentration (depending on the sign of the trap) and the total doping concentration. This affects models that depend on acceptor, donor, and total doping concentrations such as mobility and lifetime. However, this option has no effect on *mobility calculations* if a `MobilityDoping` file is used (see [Mobility Doping File on page 465](#)) or if incomplete ionization-dependent mobility is used (see [Incomplete Ionization-Dependent Mobility Models on page 463](#)).

If you specify `Add2TotalDoping(ChargedTraps)`, then the *charged* trap concentration is added to the acceptor or donor doping concentration *for mobility calculations only*. However, this option has no effect if a `MobilityDoping` file is used (see [Mobility Doping File on page 465](#)) or if incomplete ionization-dependent mobility is used (see [Incomplete Ionization-Dependent Mobility Models on page 463](#)).

---

## Trap Examples

The following example of a trap specification illustrates one donor trap level at the intrinsic energy with a concentration of  $1 \times 10^{15} \text{ cm}^{-3}$  and capture cross-sections of  $1 \times 10^{-14} \text{ cm}^2$ :

```
Traps(
    Donor Level EnergyMid=0 fromMidBandGap Conc=1e15 exsection=1e-14
    hXsection=1e-14
)
```

## Chapter 17: Traps and Fixed Charges

### Trap Models

This example shows trap specifications with exponential distributions, appropriate for a polysilicon thin-film transistor:

```
Traps(
    (eNeutral Exponential fromCondBand Conc=1e21 EnergyMid=0
     EnergySig=0.035 eXsection=1e-10 hXsection=1e-12)
    (eNeutral Exponential fromCondBand Conc=5e18 EnergyMid=0
     EnergySig=0.1 eXsection=1e-10 hXsection=1e-12)
    (hNeutral Exponential fromValBand Conc=1e21 EnergyMid=0
     EnergySig=0.035 eXsection=1e-12 hXsection=1e-10)
    (hNeutral Exponential fromValBand Conc=5e18 EnergyMid=0
     EnergySig=0.2 eXsection=1e-12 hXsection=1e-10)
)
```

---

## Trap Models

This section describes models for capture and emission rates.

---

### Trap Occupation Dynamics

The electron occupation  $f^n$  of a trap is a number between 0 and 1, and changes due to the capture and emission of electrons:

$$\frac{\partial f^n}{\partial t} = \sum_i r_i^n \quad (525)$$

$$r_i^n = (1 - f^n)c_i^n - f^n e_i^n \quad (526)$$

Here,  $c_i^n$  denotes an electron capture rate for an empty trap and  $e_i^n$  denotes an electron emission rate for a full trap. The sum in [Equation 525](#) is over all capture and emission processes. For example, the capture of an electron from the conduction band is a process distinct from the capture of an electron from the valence band.

For the stationary state, the time derivative in [Equation 525](#) vanishes. The occupation becomes:

$$f^n = \frac{c_i^n}{(c_i^n + e_i^n)} \quad (527)$$

and the net electron capture rate due to process  $k$  becomes:

$$r_k^n = \frac{c_k^n e_i^n - e_k^n c_i^n}{(c_i^n + e_i^n)} \quad (528)$$

## Chapter 17: Traps and Fixed Charges

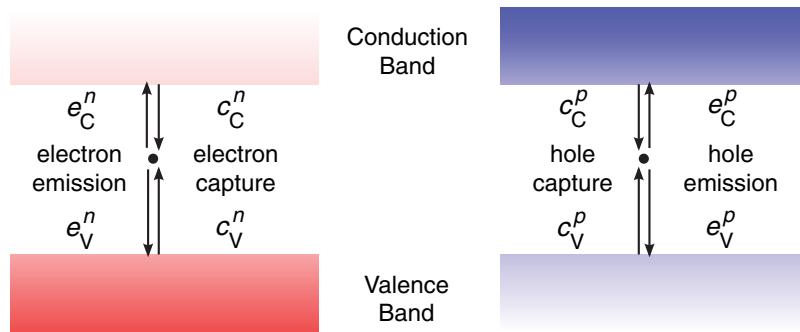
### Trap Models

The previous equations are given in the electron picture, which is the most natural one for eNeutral and Acceptor traps (see [Figure 21](#)). For hNeutral and Donor traps, you might prefer to use the equivalent hole picture.

Use the following relations to translate the electron picture into the hole picture:

- $c_i^p = e_i^n$
- $e_i^p = c_i^n$
- $f^p = 1 - f^n$
- $r_i^p = -r_i^n$

[Figure 21](#) Trap occupation dynamics: (left) electron picture and (right) hole picture



In particular, in the stationary state, for traps with concentration  $N_0$  using the V-model (see [Local Capture and Emission Rates on page 555](#)), [Equation 528](#) leads to the Shockley–Read–Hall recombination rate:

$$R_{\text{net}} = \frac{N_0 v_{\text{th}}^n v_{\text{th}}^p \sigma_n \sigma_p (np - n_{i,\text{eff}}^2)}{v_{\text{th}}^n \sigma_n (n + n_1/g_n) + v_{\text{th}}^p \sigma_p (p + p_1/g_p)} \quad (529)$$

Each capture and emission process couples the trap to a reservoir of carriers. If only a single process would be effective, in the stationary state, the trap would be in equilibrium with the reservoir for this process. This consideration (the *principle of detailed balance*) relates capture and emission rates:

$$e_i = \frac{c_i}{g} \exp \left[ \frac{E_{\text{trap}} - E_F^i}{kT_i} \right] \quad (530)$$

where:

- $E_{\text{trap}}$  is the energy of the trap.
- $E_F^i$  is the Fermi energy of the reservoir.
- $T_i$  is the temperature of the reservoir.

## Chapter 17: Traps and Fixed Charges

### Trap Models

- $g$  is the degeneracy factor. Sentaurus Device supports distinct degeneracy factors  $g_n$  and  $g_p$  for coupling to the conduction and valence bands. They default to 1 and are set with `eGfactor` and `hGfactor` in the command file, and with the `G` parameter pair in the `Traps` parameter set.

Capture and emission rates are either local (see [Local Capture and Emission Rates](#)) or nonlocal (see [Nonlocal Tunneling for Traps on page 576](#)). Sentaurus Device does not solve the current continuity equations in insulators and, therefore, supports local rates only for traps in semiconductors or at semiconductor interfaces.

Therefore, traps in insulators that are not connected to a semiconductor region by a nonlocal model do not have any capture and emission processes, and their occupation is undefined (except for `FixedCharge` traps). Sentaurus Device ignores them.

Trap occupancy computations are independent of the trap concentration. Trap occupation depends on the Fermi level and trap energy together with other parameters as described in [Local Capture and Emission Rates](#). For each trap entry in the command file, trap occupancy calculations are triggered for the region of the trap entry. After trap occupation is calculated, trap charges are computed by multiplying the trap occupation by the trap concentration.

---

## Local Capture and Emission Rates

The electron capture rate from the conduction band at the same location as the trap is:

$$c_C^n = \sigma_n \left[ (1 - g_n^J) v_{\text{th}}^n n + g_n^J \frac{J_n}{q} \right] \quad (531)$$

Similarly, the hole capture rate from the valence band is:

$$c_V^p = \sigma_p \left[ (1 - g_p^J) v_{\text{th}}^p p + g_p^J \frac{J_p}{q} \right] \quad (532)$$

Here,  $g_n^J$  and  $g_p^J$  are set with `eJfactor` and `hJfactor` in the command file, and with the `Jcoef` parameter pair in the `Traps` parameter set.  $g_n^J$  and  $g_p^J$  can take any value between 0 and 1. When they are zero (default), the V-model is used. When they are 1, the J-model (popular for modeling radiation problems) is used.

The electron emission rate to the conduction band is  $e_C^n = v_{\text{th}}^n \sigma_n \gamma_n n_1 / g_n + e_{\text{const}}^n$ .

The hole emission to the valence band is  $e_V^p = v_{\text{th}}^p \sigma_p \gamma_p p_1 / g_p + e_{\text{const}}^p$ .

For  $g_n^J = e_{\text{const}}^n = 0$  and  $g_p^J = e_{\text{const}}^p = 0$ , these rates obey the principle of detailed balance.

Above,  $n_1 = N_C \exp[(E_{\text{trap}} - E_C)/kT]$  and  $p_1 = N_V \exp[(E_V - E_{\text{trap}})/kT]$ . For Fermi statistics or with quantization (see [Chapter 14 on page 348](#)),  $\gamma_n$  and  $\gamma_p$  are given by [Equation 47](#) and [Equation 48](#) (see [Fermi Statistics on page 233](#)); otherwise,  $\gamma_n = \gamma_p = 1$ .

## Chapter 17: Traps and Fixed Charges

### Trap Models

$v_{\text{th}}^n$  and  $v_{\text{th}}^p$  are the thermal velocities. Sentaurus Device offers two options, selected by the `VthFormula` parameter pair in the `Traps` parameter set (default is 1):

$$v_{\text{th}}^{n,p} = \begin{cases} v_0^{n,p} \sqrt{\frac{T}{300\text{K}}} & \text{VthFormula=1} \\ \sqrt{\frac{3kT}{m_{n,p}(300\text{K})}} & \text{VthFormula=2} \end{cases} \quad (533)$$

$v_0^{n,p}$  are given by the `vth` parameter pair in the `Traps` section of the parameter file.

Sentaurus Device offers several options for the cross sections  $\sigma_n$  and  $\sigma_p$ . In any case, trap cross sections are derived from the constant cross sections  $\sigma_n^0$  and  $\sigma_p^0$ , which are set by the `eXsection` and `hXsection` keywords in the command file, and by the `Xsec` parameter pair in the `Traps` parameter set.

By default, the cross sections are constant:  $\sigma_n = \sigma_n^0$  and  $\sigma_p = \sigma_p^0$ .

$e_{\text{const}}^n$  and  $e_{\text{const}}^p$  represent the constant emission rate terms for carrier emission from the trap level to the conduction and valence bands. They can be set by `eConstEmissionRate` and `hConstEmissionRate` in the command file, and by the `ConstEmissionRate` parameter pair in the `Traps` parameter set (default  $e_{\text{const}}^n = 0 \text{ s}^{-1}$ ,  $e_{\text{const}}^p = 0 \text{ s}^{-1}$ ).

The `SimpleCapt` option restricts the field enhancement of cross sections to the emission rates only:

- For the computation of  $e_C^n$  and  $e_V^n$ ,  $\sigma_n$  and  $\sigma_p$  are used.
- For the computation of  $c_C^n$  and  $c_V^n$ ,  $\sigma_n^0$  and  $\sigma_p^0$  are used.

#### Note:

Using different cross sections for capture and emission rates violates the principle of detailed balance. By default, the same cross sections are used for capture and emission rates.

## J-Model for Cross Sections

This model is selected by specifying `ElectricField` in the command file. It is intended for use with the J-model and reads:

$$\sigma_{n,p} = \sigma_{n,p}^0 \left( 1 + a_1 \left| \frac{F}{1\text{Vm}^{-1}} \right|^{p_1} + a_2 \left| \frac{F}{1\text{Vm}^{-1}} \right|^{p_2} \right)^{p_0} \quad (534)$$

where  $a_1$ ,  $a_2$ ,  $p_0$ ,  $p_1$ , and  $p_2$  are adjustable parameters available as parameter pairs `a1`, `a2`, `p0`, `p1`, and `p2` in the `Traps` parameter set.

Using the `SimpleCapt` option restricts the field enhancement of cross sections to the emission rates only.

## Hurkx Model for Cross Sections

This model is selected by specifying `Tunneling(Hurkx)` in the command file. The cross sections are obtained from  $\sigma_n^0$  and  $\sigma_p^0$  using [Equation 415 on page 480](#).

Using the `SimpleCapt` option restricts the field enhancement of cross sections to the emission rates only.

## Poole–Frenkel Model for Cross Sections

The Poole–Frenkel model [1] is frequently used to interpret transport effects in dielectrics and amorphous films. The model predicts an enhanced emission probability  $\Gamma_{\text{pf}}$  for charged trap centers where the potential barrier is reduced because of the high external electric field.

To select the model, specify `PooleFrenkel` in the `Traps` specification (in the `Physics` section) of the command file.

Using the `SimpleCapt` option restricts the field enhancement of cross sections to the emission rates only.

In the Poole–Frenkel model:

$$\begin{aligned}\sigma_{n,p}^{\text{enh}} &= \sigma_{n,p}^0(1 + \text{ActivatePF} \times \Gamma_{\text{pf}} + \text{ActivatePhonon} \times \Gamma_{\text{ph}}) \\ \Gamma_{\text{pf}} &= \frac{1}{\alpha^2}[1 + (\alpha - 1)\exp(\alpha)] - \frac{1}{2} \\ \alpha &= \frac{1}{kT} \sqrt{\frac{q^3 F}{\pi \epsilon_{\text{pf}}}}^{\text{a\_power}}.\end{aligned}\quad (535)$$

and  $\Gamma_{\text{ph}}$  is an enhancement factor derived from a semiclassical theory of phonon-assisted tunneling [2]:

$$\Gamma_{\text{ph}} = \exp \left[ \frac{F^2}{E_c^2} \right]^{\text{b\_power}} \quad (536)$$

where:

- $E_c^2 = \frac{3m^* \hbar}{e^2 \tau_2^3}$  and  $\tau_2 = \frac{\hbar}{2k_B T} + \tau_1$ .
- $\tau_1$  can take negative values up to the limit of  $\tau_2 > 0$ .

## Chapter 17: Traps and Fixed Charges

### Trap Models

For this model:

- If you specify PooleFrenkel without options, then:
  - For Donor and hNeutral traps,  $\sigma_n = \sigma_n^{\text{enh}}$  and  $\sigma_p = \sigma_p^0$ .
  - For Acceptor and eNeutral traps,  $\sigma_p = \sigma_p^{\text{enh}}$  and  $\sigma_n = \sigma_n^0$ .
- If you specify PooleFrenkel(Electron), then  $\sigma_n = \sigma_n^{\text{enh}}$  irrespective of trap type.
- If you specify PooleFrenkel(Hole), then  $\sigma_p = \sigma_p^{\text{enh}}$  irrespective of trap type.

You can use PooleFrenkel(Electron) and PooleFrenkel(Hole) together to activate the model for exchange with conduction and valence bands simultaneously.

- $\epsilon_{\text{pf}}$  is an adjustable parameter. It is set by the `epsPF` parameter pair in the PooleFrenkel parameter set.
- A general power law (`a_power`) is applied to  $\alpha$  and can be set in the PooleFrenkel parameter set.
- The `b_power`,  $\tau_1$  (`ph_tau1`), and  $m^*$  (`ph_mass`) of the phonon-enhanced factor can be set in the PooleFrenkel parameter set.
- `ActivatePF` and `ActivePhonon` are scaling and activating factors for the various components of the Poole–Frenkel enhancement, and these can be set in the PooleFrenkel parameter set.

The following table lists the default values for the Poole–Frankel model parameters.

*Table 100 Parameters and their default values for Poole–Frenkel model*

Symbol	Parameter name	Electrons	Holes	Unit
<code>ActivatePF</code>	<code>ActivatePF</code>	1.0	1.0	1
<code>ActivatePhonon</code>	<code>ActivatePhonon</code>	0.0	0.0	1
$\epsilon_{\text{pf}}$	<code>epsPF</code>	11.7	11.7	1
$a_{\text{power}}$	<code>a_power</code>	1	1	1
$b_{\text{power}}$	<code>b_power</code>	1	1	1
$m^*$	<code>ph_mass</code>	0.25	0.4	1
$\tau_1$	<code>ph_tau1</code>	5e-15	5e-15	s

## Chapter 17: Traps and Fixed Charges

### Trap Models

## Makram-Ebeid–Lannoo Model

The Makram-Ebeid–Lannoo model [3] is a phonon-assisted tunnel emission model for carriers trapped on deep semiconductor levels, with its main application in two-band charge transport in silicon nitride [4]. In this model, the deep trap acts as an oscillator or a core embedded in the nitride lattice, attracting electrons (electron trap) or holes (hole trap). The deep trap is defined by the phonon energy  $W_{\text{ph}}$ , the thermal energy  $W_T$ , and the optical energy  $W_{\text{opt}}$ .

The trap ionization rate is given by:

$$P = \frac{\exp\left[\frac{nW_{\text{ph}}}{2kT} - S \coth\left(\frac{W_{\text{ph}}}{2kT}\right)\right] I_n \frac{S}{\sinh(W_{\text{ph}}/(2kT))}}{\int_{-\infty}^{\infty} P_i(W_T + nW_{\text{ph}})} \quad (537)$$

$$P_i(W) = \frac{eF}{2\sqrt{2mW}} \exp\left(-\frac{4\sqrt{2m}}{3\hbar eF} W^{3/2}\right) \quad (538)$$

where:

- $I_n$  is the modified Bessel function of the order  $n$ .
- $S = (W_{\text{opt}} - W_T)/W_{\text{ph}}$ .
- $P_i(W)$  is the tunnel escape rate through the triangle barrier of height  $W$ .

Sentaurus Device implements the Makram-Ebeid–Lannoo model for `eNeutral` and `hNeutral` traps with `Level` distributions. The electron emission rate to the conduction band  $e_C^n$  and the hole emission to the valence band  $e_V^p$  are the trap ionization rates described by [Equation 537](#) with the corresponding  $W_{\text{ph}}$ ,  $W_T$ , and  $W_{\text{opt}}$  defining the associated deep trap.

The electron capture rate from the conduction band  $c_C^n$  and the hole capture rate from the valence band  $c_V^p$  are computed based on user selection. They can be obtained from the emission rates by the principle of detailed balance or using the constant capture cross-sections:  $c_C^n = \sigma_n^0 [(1 - g_n^J) v_{\text{th}}^n n + g_n^J \frac{J}{q}]$ ,  $c_V^p = \sigma_p^0 [(1 - g_p^J) v_{\text{th}}^p p + g_p^J \frac{J}{q}]$ .

To select the model, specify `Makram-Ebeid` in the command file or the `Traps` parameter set:

- If you specify `Makram-Ebeid` without options, the trap couples to both the conduction and valence bands through Makram-Ebeid–Lannoo trap emission rates.
- If you specify `Makram-Ebeid(electron)`, an `eNeutral` trap couples to the conduction band through the Makram-Ebeid–Lannoo emission rate and to the valence band using the default emission rate.
- If you specify `Makram-Ebeid(hole)`, an `hNeutral` trap couples to the valence band through the Makram-Ebeid–Lannoo emission rate and to the conduction band using the default emission rate.

## Chapter 17: Traps and Fixed Charges

### Trap Models

The following example couples eNeutral traps with Level distributions described by the Makram–Ebeid–Lannoo model to both the conduction and valence bands:

```
Traps(
  ...
  (eNeutral Level ... Makram-Ebeid)
)
```

To couple an eNeutral trap with a Level distribution through the Makram–Ebeid–Lannoo emission rate only to the conduction band, while keeping the default emission rate to the valence band, specify the following in the command file:

```
Traps(
  ...
  (eNeutral Level ... Makram-Ebeid(electron))
)
```

By default, the electron capture rate from the conduction band to an eNeutral trap ( $e_C^n$ ) and the hole capture rate from the valence band to an hNeutral trap ( $c_V^p$ ) are computed from the corresponding emission rates using the principle of detailed balance.

By specifying Makram-Ebeid(simpleCapt), capture rates are computed as:

- $c_C^n = \sigma_n \left[ (1 - g_n^J) v_{th}^n n + g_n^J \frac{J_n}{q} \right]$  for exchange with the conduction band.
- $c_V^p = \sigma_p \left[ (1 - g_p^J) v_{th}^p p + g_p^J \frac{J_p}{q} \right]$  for exchange with the valence band.

Here,  $g_n^J$  and  $g_p^J$  are set with eJfactor and hJfactor in the command file and with the Jcoef parameter pair in the Traps parameter set.  $g_n^J$  and  $g_p^J$  can take any value between 0 and 1. When they are zero or not specified (the default), the V-model is used with the simplified rates  $c_C^n = \sigma_n^0 v_{th}^n n$  and  $c_V^p = \sigma_p^0 v_{th}^p p$ , respectively. When they are 1, the J-model is used.

You can adjust the deep trap parameters  $W_{ph}$ ,  $W_T$ , and  $W_{opt}$  in the Makram–Ebeid section of the parameter file. The default values are  $W_{ph} = 0.06\text{ eV}$ ,  $W_T = 1.4\text{ eV}$ , and  $W_{opt} = 2.8\text{ eV}$ . In addition, you can adjust the Makram–Ebeid–Lannoo tunneling masses  $m$  in the parameter file. The default value is 0.5 for both electrons and holes.

The following example changes the  $W_{ph}$ ,  $W_T$ , and  $W_{opt}$  parameters in AlN:

```
Material = "AlN" {
  Makram-Ebeid(
    mt      = 0.3 , 0.5      #[1]
    Wph    = 0.065          #[eV]
    Wt     = 1.43           #[eV]
    Wopt   = 2.7            #[eV]
  )
}
```

## Local Capture and Emission Rates From PMI

As an alternative to the previously described models, you can compute the local capture and emission rates  $c_C^n$ ,  $c_V^p$ ,  $e_C^n$ , and  $e_V^p$  directly using a PMI (see [Trap Capture and Emission Rates on page 1392](#)). Using this PMI only makes sense for traps with `Level` distributions.

---

## Trap-to-Trap Tunneling (DiscreteTrapT2T)

Sentaurus Device supports trap-to-trap tunneling between single discrete traps with all types of energetic distributions. Discrete traps of the same type can be coupled through tunneling. You can implement trap-to-trap tunneling across interfaces of regions with different materials.

Trap-to-trap tunneling is activated by specifying combinations of the following keywords in the `Physics(Region="regname"){Traps(...)}` section:

- Specify `DiscreteTrapT2T` for a trap with a discrete `Level` energy. The `Traps` statement for trap-to-trap tunneling can be defined only in a region `Physics` section, not in a `RegionInterface` or `MaterialInterface Physics` section.
- The trap energy level can be specified from `FromMidBandGap`, `FromCondBand`, or `FromValBand` with the parameter `EnergyMid` (see  $E_0$  in [Equation 522 on page 546](#)). The `Eshift` PMI is not supported for the trap-to-trap tunneling model.
- `Location` specifies the coordinates of traps involved in the trap-to-trap tunneling process. These coordinates are replaced by the coordinates of the nearest vertex, so the traps snap to this vertex. Alternatively, use `Location=(mesh)` to select all vertices in that region. However, `Location` is optional if `Randomize` is used, or if the points for this region have been defined in the `Location` of another `DiscreteTrapT2T` specification (of another region).
- `Coupled=Tunneling` means that traps are coupled through tunneling. The coupling can exist between vertices of two different regions if the distance between traps does not exceed the value of  $R_{cutoff}$ . In `DiscreteTrapT2T`, the coupling refers to two primary processes: (i) trap-to-trap tunneling between both `eNeutral` or both `hNeutral` traps, and (ii) a trap-to-trap recombination between an `eNeutral` trap and an `hNeutral` trap.

In addition, the explicit trap equation must be activated in the `Solve` section to solve for the coupled trap occupation probability. The trap-to-trap tunneling framework can also be linked to other trap emission or capture processes. For example, in the `Physics-Traps` section, you can specify `PooleFrenkel` and `eBarrierTunneling` (or `hBarrierTunneling`) together. These other capture or emission processes are additive in nature and affect the overall occupancy of the trap.

Multiple trap entries can be defined in the same region and at the same vertex. There is no restriction to mixing a `DiscreteTrapT2T` trap entry and an implicit trap entry. You can also

## Chapter 17: Traps and Fixed Charges

### Trap Models

have multiple `DiscreteTrapT2T` entries in the same region with different trap types. For example:

```
Physics(Region="region_name") {
    Traps(
        ( eNeutral DiscreteTrapT2T Gaussian ... )
        ( hNeutral DiscreteTrapT2T Uniform ... )
        ( eNeutral DiscreteTrapT2T Table(...) ... )
        ( hNeutral DiscreteTrapT2T #level traps ... )
        ( # implicit continuous traps definition 1 )
        ( # implicit continuous traps definition 2 )
    ...
)
Math{...
    Traps(Damping=0)      # Set Damping=0 for DiscreteTrapT2T
}
Plot{
    T2Tflux             # add T2Tflux to be output in TDR file
}
Solve{...
    Transient(
        InitialTime=0 FinalTime=1.0
        InitialStep=1e-7 MaxStep=1e-1 MinStep=1e-20
    ){ Coupled{ Poisson Electron Hole Traps } }
}
```

#### Note:

The `eNeutral` or Acceptor trap type caters to electrons, and the `hNeutral` or Donor trap type caters to holes. Interface traps are not supported for `DiscreteTrapT2T`; only region traps are supported.

Only `DiscreteTrapT2T` trap entries will couple with one another, and these are solved as a separate explicit trap equation coupled with the Poisson and continuity equations in the Newton scheme. Those implicit continuous trap entries are solved independently since they do not couple to each other. The indirect link between all trap entries is through the total space charge that goes into the Poisson and continuity equations. The fluxes between trap vertices can be plotted with the keyword `T2Tflux` in the `Plot` section, but this only applies to the case of `InelasticPhonon` trap-to-trap transition. The output contains `eT2Tflux` and `hT2Tflux`, which denote the electron and hole charge transport between trap vertices, respectively.

A typical `DiscreteTrapT2T` trap entry contains the following statements:

```
(
    # Define trap type and trap energy levels
    eNeutral
    FromCondBand EnergyMid=<float>
    Level
    # Gaussian EnergySig=<float>          # or Exponential, Uniform, Table
    # RandomizeEnergy=<int>                 # set seed for randomized energy
```

## Chapter 17: Traps and Fixed Charges

### Trap Models

```
# Define DiscreteTrapT2T essential keywords
DiscreteTrapT2T Coupled=Tunneling
Transition=InelasticPhonon
OutputT2TPPaths("t2t_debug_file.txt") # define only once
T2TPDErrorScale = 1e-15      # define only once in all trap entries

# Define DiscreteTrapT2T concentration
T2TconcScale=<float>
T2TfixedConc=<float>
# Conc=<float>                      # cannot use together with T2TfixedConc

# Define DiscreteTrapT2T locations
Location=(smesh)                      # choose every vertex in region
# Location=((x0 y0 z0) (x1 y1 z1) ...) # cannot use with
#                                         # Sentaurs Mesh
# Randomize=<int>
# NumberOfRandomTraps=<int>

# Local capture/emissions to Ec/Ev
eXSection= 1.5e-14 hXSection= 1.5e-14
PooleFrenkel(Electron)

# Nonlocal tunneling from trap to Ec/Ev at interfaces/contacts
eBarrierTunneling(NonLocal="NLM1" NonLocal="NLM2")
HuangRhys=70 PhononEnergy=0.05 TrapVolume=1e-1
)
```

The following are switched on automatically when any `DiscreteTrapT2T` trap entries are detected:

```
Physics { Discontinuity }
Math { Traps(RegionWiseAssembly) }
```

This means hetero-double vertices are defined automatically at region interfaces. To reconcile implicit continuous trap results with the multilevel `DiscreteTrapT2T` results, you must switch on `Discontinuity` and `RegionWiseAssembly` in your implicit trap setup.

The traps involved in the tunneling process snap to the closest vertex. Because the traps are discrete, having a good mesh in the regions where trap-to-trap tunneling occurs is important to obtain meaningful results.

All the snapped vertices of the `DiscreteTrapT2T` entries in different regions are grouped together. Then, Sentaurs Device finds all the possible tunneling paths that could form pairs of traps, between which trap-to-trap tunneling will occur. As such, trap-to-trap tunneling across interfaces of different regions (with different materials) is possible.

## Chapter 17: Traps and Fixed Charges

### Trap Models

There are different ways to set the trap concentration in `DiscreteTrapT2T`:

- By default, the local trapped carrier density is such that, in the control volume for the trap where the vertex is located, there is one trapped charge if the trap is occupied, or zero charge otherwise. To map the number of discrete traps to a concentration, the parameter `T2TconcScale` is introduced:

$$T2TconcScale = \frac{(\text{Concentration}) \times (\text{Volume of region})}{\text{Total number of region T2T traps}} \quad (539)$$

- In cases where not all the vertices of a region are chosen, it is not trivial to calculate the sum of the Voronoï volumes of all the trap-to-trap vertices. By defining `Conc=<float>` in the trap-to-trap entry, an internal trap-to-trap scaling is calculated as:

$$\text{internalT2TconcScale} = \frac{\text{Conc} \times (\text{Sum volume of T2T traps})}{\text{Total number of T2T traps}} \quad (540)$$

If you have also defined `T2TconcScale=<float>`, then the final multiplier to the trap at each vertex becomes:

$$\text{finalT2TconcScale} = \text{internalT2TconcScale} \times T2TconcScale \quad (541)$$

- Alternatively, you can fix the concentration at each trap-to-trap vertex by the keyword `T2TfixedConc=<float>`. In such a case, the number of traps at each trap-to-trap vertex is `T2TfixedConc × vertex_volume`. This definition is similar to how `Conc=<float>` is used in the implicit continuous traps.

If you have also specified `T2TconcScale=<float>`, then the final number of traps at each trap-to-trap vertex is:

$$\text{Number of traps at each vertex} = T2TfixedConc \times \text{vertex\_volume} \times T2TconcScale \quad (542)$$

A number of randomly placed traps in the region can be generated using the keywords `Randomize` and `NumberOfRandomTraps`. `Randomize` can be specified with `Location` as long as the total number of traps is not greater than the number of vertices in the region.

With trap energy distribution and trap-to-trap tunneling, this means that every trap level of the distribution can be coupled to every other level at the same vertex as well as every other trap energy level of connected neighboring vertices.

To reduce the total number of trap-to-trap paths for faster computation, you can do one of the following:

- Control the number of connections at a vertex, by using the following parameters in the `.par` file:

```
T2T_Rcutoff = <float>          # [um]
T2T_Rcutoff_inner = <float>      # [um], default of 0
```

In this case, only vertices within the distances of `[T2T_Rcutoff_inner, T2T_Rcutoff]` are valid for connection. Setting `T2T_Rcutoff_inner > 0` also means not coupling the different trap energy levels of a distribution at the same vertex.

## Chapter 17: Traps and Fixed Charges

### Trap Models

- Control the number of discrete trap levels of the distribution in the `Math` section:

```
TrapDLN(PeakSampling) = <int>    # default is 13
```

`SFactor` (using the PMI or dataset name only) is supported for `DiscreteTrapT2T` and the following example lists the allowed combinations of `SFactor` and other `DiscreteTrapT2T` keywords:

```
Physics(Region="reg_name") {
    Traps(
        # Level traps of T2T with defined T2TfixedConc
        # SFactor normalized by its region peak value
        (eNeutral DiscreteTrapT2T ...
            Level FromCondBand EnergyMid=<float>
            T2TfixedConc = <float>                      # units of 1/cm^3
            T2TconcScale = <float>
            SFactor = "pmi_name"                         # or "Dataset_name"
        )

        # Level traps of T2T with no T2TfixedConc
        # SFactor not normalized by its region peak value
        (eNeutral DiscreteTrapT2T ...
            Level FromCondBand EnergyMid=<float>
            T2TconcScale = <float>
            SFactor = "pmi_name"                         # or "Dataset_name"
            -NormalizeSFactor                           # do not normalize SFactor
        )

        # COMBINATION NOT ALLOWED: Level traps of T2T with defined Conc
        # An error will terminate the simulation if such a combination
        # is detected during syntax parsing
        (eNeutral DiscreteTrapT2T ...
            Level FromCondBand EnergyMid=<float>
            Conc = <float>
            T2TconcScale = <float>
            SFactor = "pmi_name"                         # or "Dataset_name"
        )

        # Trap energy distribution of T2T with defined Conc
        # SFactor not normalized by its region peak value
        (eNeutral DiscreteTrapT2T ...
            FromCondBand EnergyMid=<float> EnergySig=<float>
            Gaussian                                # or Exponential or Uniform or Table
            Conc = <float>                            # units of 1/cm^3/eV
            T2TconcScale = <float>
            SFactor = "pmi_name"                     # or "Dataset_name"
            -NormalizeSFactor                       # do not normalize SFactor
        )

        # Random trap energy distribution of T2T with defined Conc
        (eNeutral DiscreteTrapT2T ...
            FromCondBand EnergyMid=<float> EnergySig=<float>
            Gaussian                                # or Exponential or Uniform or Table
        )
    )
}
```

## Chapter 17: Traps and Fixed Charges

### Trap Models

```
    Conc = <float>                                # units of 1/cm^3
    RandomizeEnergy = <int>                         # set random seed
    T2TconcScale = <float>
    SFactor = "pmi_name"                            # or "Dataset_name"
)
}
}
```

Essentially, the normalized `SFactor` (normalized by the peak value of `SFactor` in the region) is multiplied by the local concentration of each vertex. To switch off the normalization of `SFactor`, use the `-NormalizeSFactor` option.

When are the `DiscreteTrapT2T` and implicit traps solved? Consider the following `Solve` section:

```
Solve {
    Poisson                                     <--- implicit solved
    Coupled {Poisson Traps}                     <--- implicit and T2T solved
    Coupled {Electron Hole}                      <--- implicit solved
    Coupled {Poisson Electron Hole Traps}       <--- implicit and T2T solved
    ...
}
```

To stop the implicit traps from being solved, freeze the traps. To not solve `DiscreteTrapT2T`, do not include `Traps` in the `Coupled` statement.

#### Note:

Even if the traps are frozen, the space charge will still contain the values of the trapped charges (`frozen_TrapOccupation`  $\times$  Concentration) of the various trap entries.

When you freeze the traps, you cannot solve `Traps` in any `Coupled` statement. Otherwise, Sentaurus Device will terminate with an error message.

You can also output all the trap-to-trap tunneling paths by specifying the keyword `OuputT2TPaths("filename.txt")` in the `Traps` section where trap-to-trap tunneling is activated. At the end of this file, a summary of the concentration for each `DiscreteTrapT2T` trap entry is printed so that you can verify the actual concentration being used.

#### Note:

When you use the explicit trap equation for `DiscreteTrapT2T`, you must set `Traps(Damping=0)` in the `Math` section.

There are different types of trap-to-trap tunneling model in Sentaurus Device:

- Mode Oscillator Model
- Inelastic Phonon Model

## Chapter 17: Traps and Fixed Charges

### Trap Models

## Mode Oscillator Model

The traps involved in trap-to-trap tunneling processes are regarded as single-mode oscillators embedded in the dielectric matrix [5] with a capture rate described by:

$$c_i = \frac{C_f \frac{\hbar W_T \sqrt{\pi}}{m_t m_0 r_{j,i}^2 Q_0 \sqrt{kT}} \exp \left( -\frac{W_{\text{opt}} - W_T}{2kT} \right) \exp \left( -\frac{2r_{j,i} \sqrt{2m_t m_0 W_T}}{\hbar} \right) \exp \left( -\frac{|E_{\text{trap}}^i - E_{\text{trap}}^j + |E_{\text{trap}}^i - E_{\text{trap}}^j|}{2kT} \right)}{f_j} \quad (543)$$

where:

- Transitions occur between localized state  $i$  with energy  $E_{\text{trap}}^i$  and neighboring localized states  $j$  with energies  $E_{\text{trap}}^j$ .
- $W_{\text{opt}}$  is the trap optical ionization energy.
- $W_T$  is the trap thermal ionization energy.
- $Q_0 = \sqrt{2(W_{\text{opt}} - W_T)}$ .
- $r_{i,j}$  is the spatial distance between traps  $i$  and  $j$  involved in the transition.
- $f_j$  is the localized trap  $j$  occupation probability.
- $C_f$  is a multiplication factor, which is 1 by default.
- $m_t$  is the trap-to-trap tunneling mass.

Tunneling between two traps is activated only if the spatial distance between them is smaller than the user-defined parameter  $R_{\text{cutoff}}$ , which limits the interaction distance for tunneling. By default, the parameter is infinite, so potentially all traps defined in all regions (specified as trap-to-trap tunneling regions) are considered. You can adjust  $R_{\text{cutoff}}$  to reduce the size of the Jacobian and numeric issues. Ideally,  $R_{\text{cutoff}}$  should be of the order of the spatial distance between traps to allow only first-order neighbor interactions.

The transition rate parameters  $W_{\text{opt}}$ ,  $W_T$ ,  $R_{\text{cutoff}}$ ,  $C_f$ , and  $m_t$  are available through the parameter file in the region trap section.

*Table 101 Parameters and their default values for trap-to-trap tunneling*

Symbol	Parameter name	Default value	Unit
$W_{\text{opt}}$	T2T_Wopt	3	eV
$W_T$	T2T_WT	1.5	eV
$R_{\text{cutoff}}$	T2T_Rcutoff	$1 \times 10^{30}$	cm
$C_f$	T2T_factor	1	1
$m_t$	T2T_m	0.5	1

## Chapter 17: Traps and Fixed Charges

### Trap Models

For trap-to-trap vertex pairs ( $v1, v2$ ) that lie across a region interface ( $reg1/reg2$ ), the trap-to-trap tunneling parameters are used in the following order:

1. If there is a RegionInterface="reg1/reg2" Traps section in the parameter file, then T2T\_Wopt, T2T\_WT, T2T\_m, and T2T\_factor of the RegionInterface parameters are used directly in the transition rate calculation. Note that in RegionInterface, T2T\_Rcutoff has no meaning.
2. Otherwise, the trap-to-trap tunneling parameters are taken as the average of those values defined in the trap-to-trap tunneling parameters of each region  $reg1$  and  $reg2$ .

## Inelastic Phonon Model

Referring to [Equation 525](#), the net change of electron trap occupation at trap  $i$ , due to capture and emission processes from  $i$  to its neighboring (connected) traps is:

$$r_i^n = \{f_i(1-f_j)c_{j,i} - (1-f_i)f_j e_{i,j}\} \quad (544)$$

where  $c_{j,i}$  is the electron capture rate from trap  $j$  to trap  $i$ , and  $e_{i,j}$  is the electron emission rate from trap  $i$  to trap  $j$ . A similar expression exists for holes.

### Note:

The trap-to-trap tunneling capture and emission terms here differ from those that describe the transport from the trap to the continuous states of the conduction or valence bands.

The inelastic phonon model for the trap-to-trap process is adapted from [\[6\]](#). You list the charge capture rate (from trap  $j$  to trap  $i$ ) as:

$$c_{j,i}(\vec{x}_i) = \frac{1}{\tau_0} T_{j,i}((\vec{x}_j|E), \vec{x}_i) M_{i,j}(l) \quad (545)$$

where  $\tau_0$  is a tuning parameter,  $T_{j,i}$  is the charge tunneling probability from trap  $j$  to trap  $i$ , and  $M_{i,j}$  is the phonon transition probability between the two trap energy levels.

The tunneling probability is computed using a simplified WKB formulation:

$$T_{j,i}((\vec{x}_j|E), \vec{x}_i) = \frac{|\Psi(x_j)|^2}{|\Psi(x_i)|^2} = \exp -\frac{2}{\hbar} \int_{x_j}^{x_i} \sqrt{2m|V(x) - E|} dx \quad (546)$$

where  $m$  is the tunneling mass, and  $V(x)$  is the conduction or valence energy band variation between the traps  $i$  and  $j$ . You further assume that the conduction or valence band energies between the traps are linear. In the case of traps that are located in different materials, such that the tunneling path crosses a material interface with discontinuities in the conduction or valence band, the tunneling probability becomes a multiplication of tunneling probabilities in each material.

## Chapter 17: Traps and Fixed Charges

### Trap Models

The phonon transition probability is:

$$M_{i,j}(l) = \left[ \alpha \frac{(l \mp S)^2}{S} + 1 - \alpha \right] \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{\chi(l+\chi)}} l \exp \left( -S(2f_B + 1) + \frac{l\hbar\omega_0}{2kT} + \chi \right) \quad (547)$$

where:

- The upper sign is for electrons and the lower sign is for holes.
- $S$  is the Huang–Rhys factor.
- $\alpha$  is a tuning factor for the phonon term, similar to [Equation 560](#).
- $z = 2S\sqrt{f_B(f_B + 1)}$  and  $\chi = \sqrt{l^2 + z^2}$ .
- $f_B = \frac{1}{\exp(\frac{\hbar\omega_0}{kT}) - 1}$  is the Bose–Einstein distribution function for phonons.
- $l = \frac{|E_{ti} - E_{tj}|}{\hbar\omega_0}$ .

The key physics are that spatial transport is modeled by the tunneling probability and inelastic transport of the trap energy difference is modeled by the phonon transition probability.

You assume that the trap occupation follows a Fermi-type distribution:

$$f_{ti} = \frac{1}{1 + g_i \exp\left(\beta \frac{E_{ti}}{kT_i}\right)} \quad (548)$$

where  $g_i$  is the degeneracy factor and  $\beta = \begin{cases} 1 & \text{for eNeutral} \\ -1 & \text{for hNeutral} \end{cases}$

The concept of detailed balance states that there should be zero net transition in each of the different processes at equilibrium, that is,  $r_i^n = 0$ .

Substituting this trap occupancy distribution ([Equation 548](#)) into the stationary condition ( $r_i^n = 0$ ), you can derive the relationship between the capture and emission terms for each trap-to-trap process as:

$$\frac{e_{i,j}}{c_{j,i}} = \frac{g_i}{g_j} \exp\left(\beta \frac{E_{ti}}{kT_i} - \frac{E_{tj}}{kT_j}\right) \quad (549)$$

Therefore, after the capture rate is computed, you can compute the emission rate using this expression.

The command file syntax for the inelastic phonon model can be defined only in a `Region-Physics` section with the keyword `Transition=InelasticPhonon`.

## Chapter 17: Traps and Fixed Charges

### Trap Models

#### Note:

You must define Transition=InelasticPhonon in the Physics sections of every region, whereby you connect the traps using this inelastic phonon trap-to-trap process.

The additions to the parameter file are:

```
Traps {
    T2T_phonon_energy = <float> , <float> # [eV] phonon energy
    T2T_tau0 = <float> , <float>           # [s] fitting parameter
    T2T_S = <float> , <float>             # [1] Huang-Rhys factor
    T2T_g = <float> , <float>             # [1] degeneracy factor
    T2T_mass = <float> , <float>       # [1] relative tunneling mass to m0
    T2T_alpha = <float> , <float>       # [1] tuning factor for phonon term
}
```

You can define these parameters only in a Region or Material section in the parameter file. Only T2T\_tau0 can be additionally defined in a RegionInterface section to allow for special fine-tuning between traps located across region interfaces. Mole fraction dependence is not supported. [Table 102 on page 571](#) lists the default values for these parameters.

## Trap-to-Trap Recombination

The trap-to-trap recombination rate assumes mutually exclusive probabilities of electron and hole tunneling to the vertices or sites of each other for recombination:

$$R_{ij} = \frac{1}{\tau_{\text{recom}}} [T_{ij(e)} \times M_{ij(e)}(l) + T_{ji(h)} \times M_{ij(h)}(l)] \quad (550)$$

where:

- The tunneling probabilities ( $T_{ij(e)}$ ,  $T_{ji(h)}$ ) and the phonon probabilities ( $M_{ij(e)}(l)$ ,  $M_{ij(h)}(l)$ ) assume the same form as the InelasticPhonon trap-to-trap tunneling capture process and use almost the same set of parameters except for a new modification of:

$$l = \frac{\beta |E_{ti} - E_{tj}|}{\hbar \omega_0} \quad (551)$$

$\beta$  has been introduced to add another degree for fitting.

- $\tau_{\text{recom}}$  is a fitting parameter.

## Chapter 17: Traps and Fixed Charges

### Trap Models

Similarly, by detailed balance, you can determine the trap-to-trap generation rate,  $G_{i,j}$ , as follows:

$$\frac{R_{i,j}}{G_{i,j}} = g_i g_j \exp \beta \frac{E_{ti}}{kT_i} - \frac{E_{tj}}{kT_j} \quad (552)$$

Trap-to-trap recombination can be activated by the keyword `T2Trecombination` in the `DiscreteTrapT2T` trap entry. The parameters of the trap-to-trap generation or recombination rates can be modified in the `Traps` section of the parameter file:

```
T2T_recom= <float>, <float> # [s] recombination rate fitting parameter
T2T_recom_beta= <float>, <float> # [1] energy fitting parameter
```

*Table 102 Default values for parameters of inelastic phonon model and trap-to-trap recombination*

Parameter	Default for eNeutral	Default for hNeutral	Unit
<code>T2T_phonon_energy</code>	0.057	0.057	eV
<code>T2T_tau0</code>	1e-4	1e-4	s
<code>T2T_S</code>	2	2	1
<code>T2T_g</code>	1	1	1
<code>T2T_mass</code>	0.1	0.5	1
<code>T2T_alpha</code>	1	1	1
<code>T2T_recom</code>	1e10 (only one value)		s
<code>T2T_recom_beta</code>	1 (only one value)		1

# Trap-to-Trap Flux of InelasticPhonon Model

To visualize the trap-to-trap inelastic phonon transport processes of charges between trap vertices, you compute an electron density vectorial flux (units of  $\text{#cm}^{-3}\text{s}^{-1}$ ) according to the formula:

$$\begin{aligned}
\bar{J}_{e, v1} &= N_{t1} \quad N_{ti}(\{c_{i, 1(n)(k)} f_{ti}(1 - f_{t1(k)}) \\
&\quad i = 2 \\
&\quad - e_{1, i(n)(k)} f_{t1(k)}(1 - f_{ti})\} \cdot \frac{\bar{v}_i - \bar{v}_1}{|\bar{v}_i - \bar{v}_1|} \boxed{\square} \\
&\quad M(hNeutral) \\
&+ N_{t1} \quad N_{tj}(\{-R_1 f_{t1(k)} f_{tj} \\
&\quad j = 2 \\
&\quad + G_{1j(k)}(1 - f_{t1(k)})(1 - f_{tj})\} \cdot \frac{\bar{v}_j - \bar{v}_1}{|\bar{v}_j - \bar{v}_1|} \boxed{\square}
\end{aligned} \tag{553}$$

A similar expression is derived for the hole flux. These fluxes contain both inelastic phonon trap-to-trap tunneling and recombination processes. To output the trap-to-trap fluxes, add the following option to the `Plot` section:

Plot { T2Tflux }

The final plotted quantities are the electron and hole vectorial fluxes (units of  $\text{#s}^{-1}$ ) in the TDR output file:

$$eT2Tflux = \bar{J}_{e,v1} \times Vol_{v1} \quad (554)$$

$$hT2Tflux = \bar{J}_{h,v1} \times Vol_{v1} \quad (555)$$

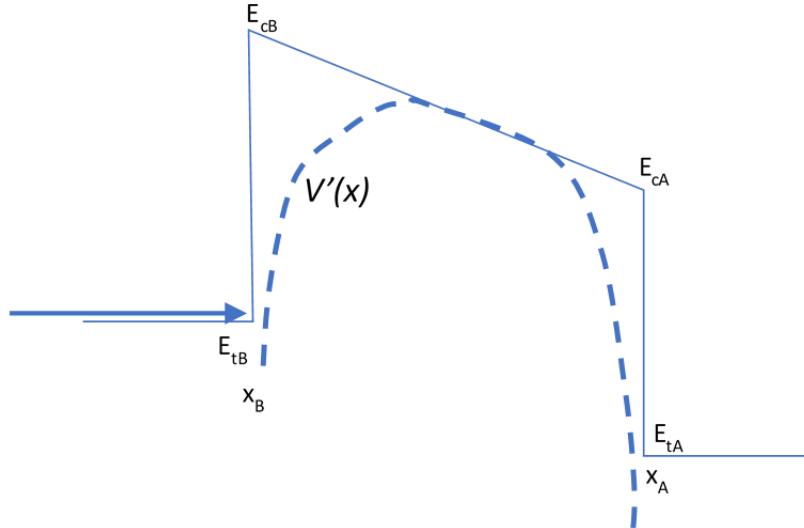
## Coulomb Potential-Induced Barrier Lowering for Inelastic Phonon Trap-to-Trap Tunneling

Traps at a locality inevitably create a Coulomb potential well, and this modifies the shape of the potential barrier between traps, potentially lowering the effective barrier of tunneling and, therefore, resulting in an increase in tunneling probability of the charges between trap locations.

## Chapter 17: Traps and Fixed Charges

### Trap Models

**Figure 22** Original barrier,  $V(x)$ , is formed by the conduction band energy at the vertices at  $x_A$  and  $x_B$ ; the Coulomb potential modifies and lowers the barrier into the blue-dashed line,  $V'(x)$



**Figure 22** shows the modification of the barrier between two traps located at  $x_A$  and  $x_B$ . The modified barrier caused by the Coulomb potential well at each trap location can be approximated by the following formula:

$$V'(x) = V(x) - \frac{k_{cA}q}{4\pi\epsilon_A(|x-x_A| + \Delta_A)} - \frac{k_{cB}q}{4\pi\epsilon_B(|x-x_B| + \Delta_B)} \quad (556)$$

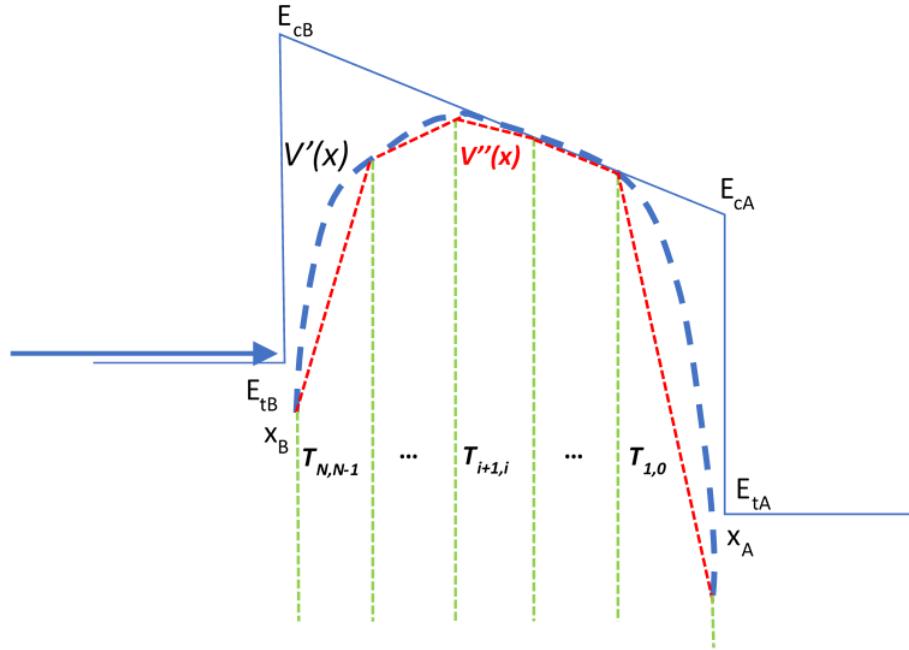
where  $V(x)$  is the original energy barrier (that is, conduction band in [Figure 22](#)), and  $k_{cA}$  and  $k_{cB}$  are dimensionless tuning factors for the potential wells from  $x_A$  and  $x_B$ , respectively. The permittivities are represented by  $\epsilon_A$  and  $\epsilon_B$ , associated with the regions defined in the trap entries of  $E_{tA}$  and  $E_{tB}$ , respectively. To avoid numeric infinity when  $x=x_A$  and  $x=x_B$ ,  $\Delta_A$  and  $\Delta_B$  are added.

To apply the WKB approximation to the tunneling probability calculation, the modified barrier is divided into  $N$  segments, whereby each segment is a linearized representation of the modified barrier, as shown in [Figure 23](#).

## Chapter 17: Traps and Fixed Charges

### Trap Models

Figure 23 Modified barrier  $V'(x)$  is divided into  $N$  linear segments,  $V''(x)$ ; the partial transmission coefficient  $T_{i+1,i}$  is computed for each segment



Assuming a linear segment between  $(x_{i+1}, x_i)$ , the transmission coefficient of this segment can be written as the 0th order WKB formulation as:

$$T_{i+1,i}(x_{i+1}, x_i) = \exp -\frac{2i}{\hbar} \int_{x_i}^{x_{i+1}} \sqrt{2m(E - V''(x))} dx \quad (557)$$

where  $V''(x)$  is a linearization of  $V'(x)$  in the interval  $(x_{i+1}, x_i)$ .

The 0th order WKB approximation is used, which is simply an exponential, so the final transmission coefficient is the product of the transmission of each segment of transmission:

$$T_{BA}(x_B, x_A) = \prod_{i=0}^{N-1} T_{i+1,i}(x_{i+1}, x_i) \quad (558)$$

The modified barrier for hole tunneling can be formulated as:

$$V'(x) = V_{\text{valence}}(x) + \frac{k_{cA}q}{4\pi\epsilon_A(|x-x_A| + \Delta_A)} + \frac{k_{cB}q}{4\pi\epsilon_B(|x-x_B| + \Delta_B)} \quad (559)$$

The rest of the analysis follows a similar fashion, so it is not shown here.

## Chapter 17: Traps and Fixed Charges

### Trap Models

#### Note:

This feature only works with the `InelasticPhonon` model for `DiscreteTrapT2T`. The cases covered include trap-to-trap transport between similar trap types and trap-to-trap recombination processes.

The activating code is contained in the command file and the parameter file:

```
Physics(Region="reg_name") {
    Traps(
        (DiscreteTrapT2T eNeutral FromCondBand Emid=<float>
            Transition=InelasticPhonon
            T2TCoulombPotential(N=<int> kc=<float> delta=<float>)
        )
        (DiscreteTrapT2T hNeutral FromValBand Emid=<float>
            Transition=InelasticPhonon
            T2TCoulombPotential(N=<int> kc=<float> delta=<float>)
        )
    )
}
```

Here, `N` is the number of segments for the modified potential barrier, `kc` is dimensionless, and `delta` is specified in micrometers. The default values are `N=5`, `kc=1`, and `delta=1e-4` μm. The epsilon value in the formulation is taken from the parameter file:

```
Epsilon {
    epsilon = <float>    # relative permittivity
}
```

When trap-to-trap transport occurs between two trap vertices that belong to different trap entries with different values of `N`, the final number of segments is chosen to be the larger `N` value between the two. In the case of trap vertices located in different regions across a region interface, the number of segments is taken to be the `N` value defined in each region.

The parameter `kc` shifts the modified potential barrier upward or downward; whereas, `delta` controls the curvature of the potential barrier at the end regions of the trap locations. In general, increasing `kc` shifts the potential barrier downward for the conduction band (and reversely for the valence band).

The smaller `delta` is, the larger `N` should be to better capture the curvature of the potential wells at the end locations of the traps. However, a larger `N` will also result in a higher computation volume and a longer simulation time.

If the trap level is shallow, then set `kc < 1`. Otherwise, larger values of `kc` can quickly shift the entire barrier below the trap level and the transmission will always be 1.0.

## Nonlocal Tunneling for Traps

Traps can be coupled to nearby interfaces and contacts by tunneling. Sentaurus Device models nonlocal tunneling to traps as the sum of an inelastic phonon-assisted process and an elastic process [7][8].

To use the nonlocal tunneling model for traps:

- For each interface or contact that you want to couple to traps by tunneling, generate a nonlocal mesh. A nonlocal mesh describes the tunneling paths between the vertices where the traps are located, and the interface or contact for which the nonlocal mesh is constructed (see [Nonlocal Meshes on page 202](#)).
- Specify `eBarrierTunneling` (for coupling to the conduction band) and `hBarrierTunneling` (for coupling to the valence band) in the trap specification in the command file. Provide the names of the nonlocal meshes that the trap must be coupled to using the `NonLocal` keyword of `eBarrierTunneling` and `hBarrierTunneling`.
- Adjust `TrapVolume`, `HuangRhys`, and `PhononEnergy`, the tunneling masses, and the interface-specific prefactors (see [Nonlocal Tunneling Parameters on page 830](#)).

**Note:**

To model trap-assisted leakage through a barrier that contains traps, at least two separate nonlocal meshes are needed, one for each side of the barrier.

For example:

```
Traps(
    hNeutral Conc=1e15 fromMidBandGap EnergyMid=0 Level TrapVolume=1e-7
    eBarrierTunneling(Nonlocal="NLM1" Nonlocal="NLM2")
)
```

## Electron Capture Rate for the Phonon-Assisted Transition

The electron capture rate for the phonon-assisted transition from the conduction band is:

$$c_{C,\text{phonon}}^n = \frac{\sqrt{m_t m_0^3 k^3 T_n^3} g_C}{\hbar^3 \sqrt{\chi}} V_T S \omega \left[ \frac{\alpha(S-l)^2}{S} + 1 - \alpha \right] \exp \left[ -S(2f_B + 1) + \frac{\Delta E}{2kT} + \chi \right] \times \frac{z}{l + \chi} F \frac{|E_{F,n} - E_C|}{kT_n} \frac{|\Psi(z_o)|^2}{|\Psi(0)|^2} \quad . \quad (560)$$

where:

- $V_T$  is the interaction volume of the trap. It is set by `TrapVolume` (in  $\mu\text{m}^3$ ) in the command file and in the `Traps` parameter set. The default is 0. `TrapVolume` must be positive when tunneling is activated.

## Chapter 17: Traps and Fixed Charges

### Trap Models

- $S$  is the Huang–Rhys factor. It is set by `HuangRhys` (dimensionless) in the command file and in the `Traps` parameter set. The default is 0.
- $\hbar\omega$  is the energy of the phonons involved in the transition. It is set by `PhononEnergy` (in eV) in the command file and in the `Traps` parameter set. The default is 0.
- $\alpha$  is a dimensionless parameter. It is set by `alpha` (dimensionless) in the command file and in the `Traps` parameter set. The default is 1.
- $l$  is the number of the phonons emitted in the transition.
- $f_B = [\exp(\hbar\omega/kT) - 1]^{-1}$  is the Bose–Einstein occupation of the phonon state.
- $z = 2S\sqrt{f_B(f_B + 1)}$ .
- $\chi = \sqrt{l^2 + z^2}$ .
- $\Delta E = E_C + 3kT_n/2 - E_{\text{trap}}$ , which is the dissipated energy.
- The Fermi energy  $E_{F,n}$  and the electron temperature  $T_n$  are obtained at the interface or contact, while the lattice temperature is obtained at the site  $z_0$  of the trap.
- $m_t$  is the relative (dimensionless) tunneling mass.
- $g_C$  is the prefactor for the Richardson constant at the interface or contact.

For details about these parameters, see [Nonlocal Tunneling Parameters on page 830](#).

## Electron Capture Rate for the Elastic Transition

The electron capture rate for the elastic transition from the conduction band is:

$$c_{C,\text{elastic}}^n = \frac{\sqrt{8m_0}m_0^{3/2}g_C}{\hbar^4\pi}\gamma V_T [E_C(z_0) - E_{\text{trap}}]^2 \Theta[E_{\text{trap}} - E_C(0)] \sqrt{E_{\text{trap}} - E_C(0)} f \frac{|E_{F,n} - E_{\text{trap}}|}{kT_n} \frac{|\Psi(z_0)|^2}{|\Psi(0)|^2} \quad (561)$$

where  $f(x) = 1/(1 + \exp(-x))$  and  $\gamma$  is a dimensionless parameter `gamma` that can be set in the command file or the `Traps` parameter set and that defaults to 1. The emission rates are obtained from the capture rates by the principle of detailed balance (see [Equation 530](#)). The hole terms are analogous to the electrons terms. However, `hBarrierTunneling` for contacts and metals is nonphysical and, therefore, ignored.

The ratio of wavefunction is obtained from  $\Gamma_{CC}$  described in [WKB Tunneling Probability on page 833](#) as:

$$\frac{|\Psi(z_0)|^2}{|\Psi(0)|^2} = \frac{v(0)}{v(z_0)} \Gamma_{CC} \quad (562)$$

where  $v$  denotes the (possibly imaginary) velocities.

To avoid singularities, for the inelastic transition with the WKB tunneling model, the velocity  $v(z_0)$  at the trap site is replaced by the thermal velocity. For the inelastic process,  $\Gamma_{CC}$  is

## Chapter 17: Traps and Fixed Charges

### Numeric Parameters for Traps

computed at the tunneling energy  $E_C + kT_n/2$  and, for elastic process, at energy  $E_{\text{trap}}$ . For the tunneling probability, the default one-band model and the `TwoBand` option are available, as described in [WKB Tunneling Probability on page 833](#).

---

## Numeric Parameters for Traps

When used with Fermi statistics, traps sometimes lead to convergence problems, especially at the beginning of a simulation when Sentaurus Device tries to find an initial solution. Often, you can resolve this problem by changing the numeric damping of the trap charge in the nonlinear Poisson equation.

To change the numeric damping, set `Damping` to a nonnegative number in the `Traps` statement in the global `Math` section. For example:

```
Math {  
    Traps(Damping=100)  
}
```

Larger values of `Damping` increase damping of the trap charge; a value of 0 deactivates damping. The default value is 10.

Depending on the particular example, increasing damping might improve or degrade convergence behavior. There are no guidelines regarding the optimal value.

At nonheterointerface vertices, bulk traps are considered only from the region with the lowest band gap. When bulk traps from other regions are important, use the `RegionWiseAssembly` option in the `Traps` statement in the global `Math` section, which properly considers bulk traps from all adjacent regions.

---

## Visualizing Traps

You can plot the concentration of trapped electrons and holes (full concentration multiplied by trap occupation) as follows:

- Specify `eTrappedCharge` in the `Plot` section to plot the concentration of trapped electrons in `eNeutral` and `Acceptor` traps and of negative fixed charges.
- Specify `hTrappedCharge` in the `Plot` section to plot the concentration of trapped holes in `hNeutral` and `Donor` traps and of positive fixed charges.

These datasets include the contribution of interface charges. To that end, Sentaurus Device converts interface densities to volume densities and, therefore, their contribution depends on the mesh spacing. To plot interface charges separately as interface densities, use `eTrappedCharge/RegionInterface` and `hTrappedCharge/RegionInterface`. In

## Chapter 17: Traps and Fixed Charges

### Visualizing Traps

Sentaurus Visual, these quantities are selected with `Int(eTrappedCharge)` and `Int(hTrappedCharge)`.

The dataset `NetTrappedCharge` is saved automatically whenever any `Traps` statement is detected in the command file. This quantity gives the net trapped charge density at every vertex:

```
NetTrappedCharge = - eTrappedCharge + hTrappedCharge
```

This means that all of the different trap entries are summed (with appropriate signs) at each vertex to compute the `NetTrappedCharge`. A convenient use of this dataset is to load it as an `SFactor` for `FixedCharge` in a later quasistationary simulation with the following syntax:

```
Physics(Region="...") {
    Traps(
        ( FixedCharge EnergyMid=0 SFactor="NetTrappedCharge" )
    )
}
```

Specify `TotalTrapConcentration` to plot the absolute value of the net bulk trap concentration. Similarly, specify `TotalTrapConcentration/RegionInterface` to plot the absolute value of the net interface trap concentration. In Sentaurus Visual, this quantity is selected with `Int(TotalTrapConcentration)`. These quantities do not account for occupancy. You can specify `eTrapConcentration` and `hTrapConcentration` to plot the absolute values of the bulk trap concentration for acceptor-type or donor-type traps, respectively.

You can also specify the `eInterfaceTrappedCharge` and `hInterfaceTrappedCharge` quantities, to plot the amount of trapped charges at interfaces in  $\text{cm}^{-2}$ . These quantities have nonzero values only at interface vertices.

Specify `TotalInterfaceTrapConcentration` to plot the absolute value of the net interface trap concentration. This is similar to `TotalTrapConcentration/RegionInterface`; however, `TotalInterfaceTrapConcentration` is stored as a bulk quantity with a nonzero value only on interfaces. You can specify `eInterfaceTrapConcentration` and `hInterfaceTrapConcentration` to plot the absolute value of the interface trap concentrations for acceptor-type or donor-type traps, respectively.

Specify `TrapConcentration` to plot the separated full bulk trap concentration that was defined in each trap entry and at each trap energy level. This applies only in the `CurrentPlot` statement. In the `CurrentPlot` statement, you can also specify `TrapConcentrationPerEntry` to plot the integrated trap concentration for each trap entry.

The entries for trap concentration per energy per trap are displayed in the current plot file as:

```
TrapConcentration_<TrapEntryName>(<TrapType>, <EnergyDistribution>)/
<EnergyLevel>
```

## Chapter 17: Traps and Fixed Charges

### Visualizing Traps

For example, the following trap entry in the command file:

```
Traps (
  (name= "A1Int" Conc= 1e11 Acceptor Uniform EnergyMid=0
   fromMidBandGap)
)
```

is displayed as `TrapConcentration_A1Int(Ac, Un)/0` to `TrapConcentration_A1Int(Ac, Le)/12`. Note that the number of energy levels is set by `TrapDLN` and defaults to 13. The TDR file displays the integrated trap concentration as `TrapConcentration_A1Int(Ac, Un)`.

#### Note:

For traps specified in insulators at vertices on interfaces to semiconductors, the charge density in the insulator parts associated with the vertices is reassigned to the semiconductor parts. This involves rescaling the densities with the ratio of the volumes of the two parts and, typically, this distorts the data for visualization.

However, the distortion has no effect on the actual solution.

Specify `TrapConcPerEntry` to plot the trap concentrations for each energy level of each trap entry specified in the command file. This also stores the integrated trap concentration (trap concentration summed over all energy levels for a given trap entry).

In plot files, the entries for trap concentration per energy per trap are displayed as:

```
TrapConcentration_<TrapEntryName>(<TrapType>, <EnergyDistribution>)-
<EnergyLevel>
```

For example, the following trap entry in the command file:

```
Traps (
  (name= "A1Int" Conc= 1e11 Acceptor Uniform EnergyMid=0
   fromMidBandGap)
)
```

results in 13 entries from `TrapConcentration_A1Int(Ac, Un)-0` to `TrapConcentration_A1Int(Ac, Le)-12`. In addition, the integrated trap concentration is stored as `TrapConcentration_A1Int(Ac, Un)`.

To plot the recombination rates for the conduction band and valence band due to traps, specify `eGapStatesRecombination` and `hGapStatesRecombination`, respectively.

Specify `TrappingRates` in the `Plot` section to see the capture and emission rates for each trap, split into local processes (as described in [Local Capture and Emission Rates](#)), inelastic tunneling processes (see [Electron Capture Rate for the Phonon-Assisted Transition](#)), and elastic tunneling processes (see [Electron Capture Rate for the Elastic Transition](#)). For `eNeutral` and `Acceptor` traps, `capture` is electron capture. For `hNeutral` and `Donor` traps, `capture` is hole capture, and likewise for `emission`.

## Chapter 17: Traps and Fixed Charges

### Visualizing Traps

In addition, you can plot the trapped carrier density and occupancy probability versus energy at positions specified in the command file. You must define the name of the plot (.plt) file in the File section using the keyword TrappedCarPlotFile. For example:

```
File {
    ...
    TrappedCarPlotFile = "itrap_trappedcar"
}
```

Plotting is activated by including the TrappedCarDistrPlot section (similar to the CurrentPlot section) in the command file:

```
TrappedCarDistrPlot {
    MaterialInterface="Silicon/Oxide" {(0.5 0)}
    ...
}
```

The positions that define where the trapped carrier density, occupancy probability, and trap density versus energies should be plotted are specified materialwise or regionwise in the TrappedCarDistrPlot section, grouped into regions, materials, region interfaces, and material interfaces.

A set of coordinates of positions in parentheses follows the region or region interface. For example:

```
TrappedCarDistrPlot {
    MaterialInterface="Silicon/Oxide" {(0.5 0)}
    RegionInterface="Region_2/Region_3" {(0.1 0.001)}
    Region="Region_1" {(0.3 0) (0 0) (-0.1 0.2)}
    Material="Silicon" {(0.27 0) (0 0.01)}
    ...
}
```

For each position defined by its coordinates, Sentaurus Device searches for the closest vertex inside the corresponding region or on the corresponding region interface. This is the actual position where plotting is performed.

The difference between user coordinates and actual coordinates is displayed in the log file for each valid position in the TrappedCarDistrPlot section in the following format:

Position(User)	ClosestVertex	PositionClosestVertex
[(2.4000,-0.1000)	718	(2.3130,-0.0500)]
[(2.5000,-0.1000)	743	(2.3500,-0.0500)]

In addition, a simplified syntax for a global position inside the device is available:

```
TrappedCarDistrPlot {
    (0.5 0)
    ...
}
```

In this case, the distribution plotting uses the closest vertex.

## Chapter 17: Traps and Fixed Charges

### Insulator Fixed Charges

Based on trap types and positions, a unique entry is created in the generated plot. The naming is <TrapType><Counter>(<position>), where <Counter> is applied when multiple traps of the same type are used. For each entry, the Energy, TrappedChargeDistribution, DistributionFunction, and TrapDensity fields are available for plotting. In addition, for each entry, eQuasiFermi and hQuasiFermi are available for plotting, which allow you to monitor the evolution of DistributionFunction relative to quasi-Fermi levels.

---

## Insulator Fixed Charges

Sentaurus Device supports fixed charges in insulators and at insulator interfaces. Insulator fixed charges are defined with the keyword `FixedCharge` in the `Physics-Traps` section:

```
Physics (Material="Oxide") {
    Traps(
        (FixedCharge
            SpatialShape = [Gaussian | Uniform]
            Conc = <float>                      # [cm-3]
            SpaceMid = <vector>                  # [um]
            SpaceSig = <vector>                  # [um]
        )
    )
}
```

This syntax applies similarly to `MaterialInterface`, `Region`, and `RegionInterface`.

The keyword `Conc` specifies the maximum bulk or interface concentration of fixed charges in  $\text{cm}^{-3}$  or  $\text{cm}^{-2}$ , respectively. The `SpaceMid` and `SpaceSig` keywords have the same meaning as for traps (see [Energetic and Spatial Distribution of Traps on page 544](#)). They are optional for uniform spatial distributions but mandatory for Gaussian spatial distributions.

---

## References

- [1] L. Colalongo *et al.*, “Numerical Analysis of Poly-TFTs Under Off Conditions,” *Solid-State Electronics*, vol. 41, no. 4, pp. 627–633, 1997.
- [2] S. D. Ganichev *et al.*, “Distinction between the Poole-Frenkel and tunneling models of electric-field-stimulated carrier emission from deep levels in semiconductors,” *Physical Review B*, vol. 61, no. 15, pp. 10361–10365, 2000.
- [3] S. Makram-Ebeid and M. Lannoo, “Quantum model for phonon-assisted tunnel ionization of deep levels in a semiconductor,” *Physical Review B*, vol. 25, no. 10, pp. 6406–6424, 1982.

## Chapter 17: Traps and Fixed Charges

### References

- [4] K. A. Nasyrov *et al.*, “Two-bands charge transport in silicon nitride due to phonon-assisted trap ionization,” *Journal of Applied Physics*, vol. 96, no. 8, pp. 4293–4296, 2004.
- [5] K. A. Nasyrov and V. A. Gritsenko, “Charge transport in dielectrics via tunneling between traps,” *Journal of Applied Physics*, vol. 109, no. 9, p. 093705, 2011.
- [6] A. Schenk, *Advanced Physical Models for Silicon Device Simulation*, Wien: Springer, 1998.
- [7] A. Palma *et al.*, “Quantum two-dimensional calculation of time constants of random telegraph signals in metal-oxide–semiconductor structures,” *Physical Review B*, vol. 56, no. 15, pp. 9565–9574, 1997.
- [8] F. Jiménez-Molinos *et al.*, “Direct and trap-assisted elastic tunneling through ultrathin gate oxides,” *Journal of Applied Physics*, vol. 91, no. 8, pp. 5116–5124, 2002.

# 18

## Phase and State Transitions

---

This chapter presents a framework for the simulation of local phase or state transitions.

State transitions appear in device physics in various forms. Typical examples are the charge traps as described in [Chapter 17 on page 543](#). In phase-change memory (PCM) devices, different phases (for example, crystalline and amorphous) of chalcogenides are used to store information and can be modeled with the framework described here. Furthermore, in the hydrogen transport degradation model (see [MSC–Hydrogen Transport Degradation Model on page 611](#)), diffusing mobile hydrogen species might be trapped in localized states.

In this chapter, a general modeling framework called *multistate configuration* (MSC) is presented to describe transitions between phases or states. The framework allows the specification of an arbitrary number of states that interact locally by an arbitrary number of transitions. The states can be charged and carry hydrogen atoms. Transitions between two states might interact with the conduction and valence band, or with hydrogen diffusion equations to preserve charge and the number of hydrogen atoms. The structure of transitions is limited to a linear local dependency between the state occupation rates. However, arbitrary nonlinear local dependency on the solution variables of the transport equations are allowed using PMI models. The state occupation rates are solved self-consistently with the transport model both for stationary and dynamic characteristics.

---

### Multistate Configurations and Their Dynamic

A multistate configuration (MSC) is defined by the number of states  $N$  and the state occupation probabilities  $s_1, \dots, s_N$  satisfying the condition:

$$\sum_i s_i = 1 \quad (563)$$

For two states  $i$  and  $j$ , an arbitrary number of transitions (described by capture and emission rates) is allowed. The dynamic equation is then given by:

$$\dot{s}_i = \sum_{j \neq i} \sum_{t \in T_{ij}} c_{ij} s_j - e_{ij} s_i \quad (564)$$

where  $T_{ij}$  is the set of transitions between  $i$  and  $j$ . For such a transition  $t \in T_{ij}$  with capture and emission rates  $c$  and  $e$ , respectively, you have  $c = c_{ij} = e_{ji}$  and  $e = e_{ij} = c_{ji}$  if  $i$  and  $j$  denote the reference state and interacting state, respectively. The problem can be written in the compact form:

$$\dot{s} = Ts \quad (565)$$

where  $T$  is the total transition matrix, composed of the individual transition matrices.

Each state can carry a number  $K^Q$  of (positive) charges and a number  $K^H$  of hydrogen atoms (both numbers can be positive or negative, and are zero by default). Transitions between two states must satisfy conservation laws for both quantities. Required particles can be taken from several reservoirs. Reservoir particles are characterized by the corresponding numbers  $K_r^Q$  and  $K_r^H$ , and transitions specify the number  $P_r$  of involved reservoir particles. The conservation laws then read:

$$K_i - K_j = \sum_r P_r K_r \quad (566)$$

where  $K_i$  and  $K_j$  are the characteristic numbers for the reference and interacting states of the transition, respectively. The sum is taken over all reservoirs involved in the transition.

[Table 103](#) lists the available particle reservoirs and their characteristics. The reservoirs of hydrogen atoms, molecules, and ions represent the corresponding mobile species in the hydrogen transport degradation model. Their use is illustrated in [Reactions With Multistate Configurations on page 616](#).

The resulting space charge and recombination terms with the corresponding equations are taken into account automatically.

*Table 103 MSC particle reservoirs and their characteristics*

Description	Identifying string	Particle	Charge $K^Q$	Hydrogen $K^H$	Equation
Conduction band	CB	electron	-1	0	Electron
Valence band	VB	hole	1	0	Hole
Hydrogen atoms	HydrogenAtom	$H$	0	1	HydrogenAtom
Hydrogen molecules	HydrogenMolecule	$H_2$	0	2	HydrogenMolecule
Hydrogen ions	HydrogenIon	$H^+$	1	1	HydrogenIon
Hydrogen species A	HydrogenSpeciesA	H	0	1	HydrogenSpeciesA

## Chapter 18: Phase and State Transitions

### Specifying Multistate Configurations

Table 103 MSC particle reservoirs and their characteristics (Continued)

Description	Identifying string	Particle	Charge $K^Q$	Hydrogen $K^H$	Equation
Hydrogen species B	HydrogenSpeciesB	H	0	1	HydrogenSpeciesB
Hydrogen species C	HydrogenSpeciesC	H	0	1	HydrogenSpeciesC

## Specifying Multistate Configurations

A multistate configuration is specified by an `MSConfig` section placed into an `MSConfigs` section of a (region or material or global or interface) `Physics` section. It is described by its states (at least two) and transitions (each state must be involved in at least one transition) using the keywords `State` and `Transition`. An arbitrary number of `MSConfig` sections is allowed.

```
Physics ( Region = "si" ) {
    MSConfigs (
        MSConfig ( Name = "ca"
            State ( Name = "c" ) State ( Name = "a" )
            Transition ( Name = "t1"
                To="c" From="a" CEModel( "pmi_ce_msc" 0 ) )
        )
        ...
    )
}
```

This example specifies a multistate configuration with two states and one transition between them. See [Table 317 on page 1691](#) for the parameters supported by `MSConfig`.

`State` requires a `Name` as an identifier. The state can carry both a number of positive charges by specifying `Charge`, and a number of hydrogen atoms by using `Hydrogen`.

`Transition` requires several parameters. The keyword `CEModel` specifies a transition model including its optional model index (see [Transition Models on page 588](#)). The reference and interacting states of the transition are selected by the keywords `To` and `From`, respectively. A `Name` specification is mandatory as well.

A transition between differently charge states (correspondingly for hydrogen atoms) requires additional charged particles to preserve the total charge. With `Reservoirs`, you can specify a list of reservoirs (`CB` and `VB` for the conduction band and valence band, respectively), which provides the necessary number of particles specified by `Particles` as an argument to the reservoir. The conduction band serves as an electron reservoir, and the valence band is a hole reservoir.

---

## Multistate Configurations on Interfaces

You can also define MSCs on interfaces. Therefore, an MSC might look like the following example:

```
Physics (MaterialInterface="Oxide/Silicon") {
    MSConfigs (
        MSConfig (
            Name="msc0"
            State(Name="s0" Charge=0 Hydrogen=1)
            State(Name="s1" Charge=1 Hydrogen=0)
            Transition (
                Name="Depassivation" CEModel("CEModel_PMI",1)
                To="s1" From="s0"
                Reservoirs("VB"(Particles=+1)
                           "HydrogenAtom"(Particles=-1))
                FieldFromInsulator
            )
        )
    )
}
```

Observe that, for interface MSCs, the reservoirs refer to bulk reservoirs. At interfaces between the insulator and semiconductor, you can explicitly require that the field from the insulator is selected by using `FieldFromInsulator` in the MSC specification.

---

## Additional Remarks

An `eNeutral` level trap of the form:

```
(Name="eN" eNeutral Conc=1.e+13 CBRate=( "pmi_ce0" 0 )
   VBRate= ( "pmi_ce0" 1))
```

can be specified equivalently as an MSC in the following way:

```
MSConfig ( Name="eN" Conc=1.e+13
    State ( Name="s0" Charge=0 ) State ( Name="s1" Charge=-1 )
    Transition ( Name="tCB" CEModel("pmi_ce0" 0)
                  To="s1" From="s0" Reservoirs("CB"(Particles=+1)))
    Transition ( Name="tVB" CEModel("pmi_ce0" 1)
                  To="s0" From="s1" Reservoirs("VB"(Particles=+1)))
)
```

For all multistate configurations, the dynamic is always solved implicitly, that is, no extra equation needs to be specified as an argument to the `Coupled` or `Transient solve` statements.

**Note:**

Only quasistationary and transient simulations support multistate configurations. Small-signal analysis (see [Small-Signal AC Analysis on page 149](#)), harmonic balance analysis (see [Harmonic Balance on page 153](#)), and noise analysis (see [Chapter 23 on page 781](#)) do not support multistate configurations.

The computation of state occupation is influenced by the `TrapFilling` option of the `Set` and `Unset` statements in the `Solve` section (see [Table 337 on page 1715](#)). It is frozen if the trap-filling option `Frozen` is set; otherwise, the occupation rates are treated as free. You can release the frozen status again by the `Unset(TrapFilling)` statement.

The transition dynamic might become numerically unstable, that is, it cannot be solved with sufficient accuracy if, for example, the forward and backward rates of all transitions at one operation point differ by several orders of magnitude. Using `-Elimination` in `MSConfig` applies a different solving algorithm, often improving the numeric robustness in such cases.

The state occupation probabilities can be plotted in groups or individually by specifying `MSConfig` (see [Table 335 on page 1713](#)) in the `Plot` section.

---

## Transition Models

The MSC framework allows an arbitrary number of transitions between two states of an MSC. Each transition model can be either the model `pmi_ce_msc`, or a trap capture and emission PMI (see [Trap Capture and Emission Rates on page 1392](#)).

---

### The `pmi_ce_msc` Model

The `pmi_ce_msc` transition model supports the following features:

- Arbitrary number of states and transitions
- Charge states
- Several transition rate models (nucleation, growth, electron and hole exchange)
- Equilibrium computation (necessary for detailed balance processes)
- Energy and particle exchange

## States

The states are described by a base energy  $E_i$ , a degeneracy factor  $g_i$ , the number of negative elementary charges  $K_i$  (an arbitrary integer), and the energy  $E_i^-$  of one electron in the state. The inner energy of the state is then:

$$H_i = E_i + K_i E_i^- \quad (567)$$

The electron energy is the sum of the valence band energy and the user-specified value  $E_{i, \text{user}}$ .

## Equilibrium

The equilibrium occupation probabilities  $s^*$  are needed to guarantee the detailed balance principle for all transitions. The equilibrium is determined by the state parameters, the temperature, and the Fermi levels of the involved particle reservoirs. You have:

$$Z_i = g_i \exp(-\beta(H_i - K_i E_F)) \quad (568)$$

$$Z = \sum_i Z_i \quad (569)$$

$$s_i^* = Z_i / Z \quad (570)$$

Here,  $E_F$  is the quasi-Fermi energy and  $\beta$  is the thermodynamic beta. The quasi-Fermi energy is approximated inside the PMI. Let the (approximated) intrinsic density  $n_I$  and the intrinsic Fermi energy  $E_I$  be:

$$n_I = \sqrt{N_C N_V \exp(-\beta E_g)} \quad (571)$$

$$E_I = \frac{1}{2} \left[ (E_C + E_V) + kT \ln \left( \frac{N_V}{N_C} \right) \right] \quad (572)$$

Then, the carrier quasi-Fermi energies are approximated from the carrier densities by:

$$E_{F,n} = E_I + kT \ln \left( \frac{n}{n_I} \right) \quad (573)$$

$$E_{F,p} = E_I - kT \ln \left( \frac{p}{n_I} \right) \quad (574)$$

and equilibrium Fermi energy then as:

$$E_F = \frac{1}{2}(E_{F,n} + E_{F,p}) \quad (575)$$

## Transitions

You can select the transition models, listed in [Table 104](#), by using the `Formula` parameter in the parameter file.

## Chapter 18: Phase and State Transitions

### Transition Models

In this section,  $i$  denotes the *to* state, while  $j$  specifies the *from* state of the transition. For most of the models, only the forward reaction rate (capture) is given, while the backward reaction rate (emission) is computed by:

$$\frac{e}{c} = \frac{s_j^*}{s_i^*} \quad (576)$$

if not stated otherwise.

*Table 104 The pmi\_ce\_msc transition models*

Description	Formula
Arrhenius law	0
Nucleation according to Peng	1
Growth according to Peng	2
Single trap	7
MSC trap	8

#### Arrhenius Law (Formula=0)

The forward reaction rate (capture) is given by:

$$c = r_0 \exp(-\beta E_{\text{act}}) \quad (577)$$

where  $r_0$  is the maximal transition frequency and  $E_{\text{act}}$  is the activation energy.

#### Nucleation According to Peng (Formula=1)

A nucleation model, in analogy to the model in [1], [2], and [3], is given by:

$$c_N = r_0 \exp(-\beta(E_{\text{act}} + \Delta G^*(T))) \quad (578)$$

where:

$$\Delta G^*(T) = \frac{16\pi}{3} \frac{\gamma_{\text{SL}}^3}{\Delta G(T)^2} \quad (579)$$

$$\begin{aligned} \Delta G(T) &= \begin{cases} \overset{\circ}{\Delta H_2} \left[ 1 - \frac{T}{T_g} \right] 1 - \frac{\Delta H_1 T_m - T_g}{\Delta H_2 T_m} & \text{if } T < T_g \\ \overset{\circ}{\Delta H_1} \frac{T_m - T}{T_m} & \text{if } T_g < T < T_m \\ 0 & \text{if } T_m < T \end{cases} \end{aligned} \quad (580)$$

### Growth According to Peng (Formula=2)

Growing crystalline phases is often described by a growth velocity, for example, in [1]. Some of the growing model has been adopted in the following local model, which reads as:

$$c_G = r_0[1 - \exp(-\beta\Delta G(T)v_{MM})]\exp(-\beta E_{act}) \quad (581)$$

if  $T < T_m$ ; otherwise, it is zero.

### Single-Trap Transition (Formula=7)

This model depends on the carrier density  $d_c$ . It resembles some standard trap models and is intended to be used in two-state MSCs only. If the number of electrons in the reference state is greater than in the interacting state, that is,  $K_i > K_j$ , then the capture process depends on the electron density and uses  $d_c = n$ . If  $K_i < K_j$ , then  $d_c = p$ ; otherwise, you use  $d_c = 0$ . The capture rate is then given by:

$$c = \sigma v_{th} d_c \quad (582)$$

where  $\sigma$  is the cross section and  $v_{th}$  is the thermal velocity. The emission rate for electron capturing is computed as:

$$e = c \exp(\beta(E_T - E_{F,n})) \quad (583)$$

where  $E_T$  is the trap energy. For hole-capturing, the emission rate reads as:

$$e = c \exp(\beta(E_{F,p} - E_T)) \quad (584)$$

### MSC Trap With Emulated Detailed Balance (Formula=8)

The actual model generalizes the single-trapping model (Formula=7) to general MSC states and transitions. The capture rate is computed as in the single trapping case above. The emission rate, however, is computed as follows. Let  $N_n$  and  $N_p$  be the number of electrons and holes, respectively, which are destroyed during the process. Then, you have:

$$N_n - N_p = K_i - K_j \quad (585)$$

and you require:

$$\frac{e}{c} = g_{ji} \exp(-N_n \beta_n (E_{F,n} - \bar{E}_C) + N_p \beta_p (E_{F,p} - \bar{E}_V) + \beta_l (-N_n \bar{E}_C + N_p \bar{E}_V - H_{ji})) \quad (586)$$

where you used  $g_{ji} = g_j/g_i$  and  $H_{ji} = H_j - H_i$ .

The average conduction band and valence band energies are:

$$\bar{E}_C = E_C + \frac{3}{2}kT_n \quad \text{and} \quad \bar{E}_V = E_V - \frac{3}{2}kT_p \quad (587)$$

In thermal equilibrium, the detailed balance principle is satisfied.

## Model Parameters

The model requires parameters for all states and transitions to allow a consistent computation of the thermal equilibrium of the MSC as a whole. Therefore, parameters are grouped into global, state, and transition parameters.

The parameters `nb_states` and `nb_transitions` determine the number of states and transitions, respectively, and must be both consistent with the command file specification. The parameter `sindex<int>` specifies for the `<int>`-th state a parameter index, which determines a prefix for the state parameters. For example, given `sindex0=5`, the parameters prefixed with `s5_` are read for the 0-th state. A similar parameter index selection is available for transitions using the parameters `tindex<int>` (for example, `tindex2=7` reads the parameters prefixed by `t7_`).

For transitions, the specified parameter index must coincide with the model index of the transition in the command file.

*Table 105 Global parameters*

Name	Symbol	Default	Unit	Range	Description
plot	—	0	—	{0,1}	Plots some properties to screen
nb_states	—	0	—	$\geq 0$ , integer	Number of states
nb_transitions	—	0	—	$\geq 0$ , integer	Number of transitions
<code>sindex&lt;int&gt;</code>	—	—	—	$\geq 0$ , integer	Parameter index of <code>&lt;int&gt;</code> -th state
<code>tindex&lt;int&gt;</code>	—	—	—		Parameter index of <code>&lt;int&gt;</code> -th transition

*Table 106 Global material and default transition parameters of pmi\_ce\_msc*

Name	Symbol	Default	Unit	Range	Description
<code>E_g</code>	$E_g$	0.5	eV	$>0.$	Band gap
<code>N_C</code>	$N_C$	$1 \times 10^{19}$	$\text{cm}^{-3}$	$>0.$	Electron density-of-states
<code>N_V</code>	$N_V$	$1 \times 10^{19}$	$\text{cm}^{-3}$	$>0.$	Hole density-of-states
<code>T_m</code>	$T_m$	$1 \times 10^{100}$	K	$>0.$	Melting temperature
<code>T_g</code>	$T_g$	$1 \times 10^{100}$	K	$>0.$	Glass temperature

**Chapter 18: Phase and State Transitions**  
 Transition Models

*Table 106 Global material and default transition parameters of pmi\_ce\_msc (Continued)*

Name	Symbol	Default	Unit	Range	Description
DeltaH1	$\Delta H_1$	0.	J/cm <sup>3</sup>	>0.	Heat of solid-to-liquid
DeltaH2	$\Delta H_2$	0.	J/cm <sup>3</sup>	>0.	Heat of amorphous-to-crystalline
GammaSL	$\gamma_{SL}$	0.	J/cm <sup>2</sup>	>0.	Interfacial free-energy density
MM_volume	$v_{MM}$	0.	cm <sup>3</sup>	>0.	Monomer volume
Reference_E_T	—	0	—	{0,1,2}	Reference energy for $E_T$

*Table 107 Reference\_E\_T interpretation*

Value	Symbol	Description
0	$E_T = (E_C + E_V)/2 + E_{T, user}$	Midgap
1	$E_T = E_C - E_{T, user}$	Conduction band
2	$E_T = E_V + E_{T, user}$	Valence band

*Table 108 State parameters*

Name	Symbol	Default	Unit	Range	Description
E	$E_i$	0.	eV	real	Constant base energy
g	$g_i$	1.	1	>0.	Degeneracy
charge	$-K_i$	0	1	integer	Number of positive elementary charges
E_charge	$E_{i, user}^+$	0.	eV	real	Particle energy with respect to valence band
E_nb_Tpairs	—	0	—	$\geq 0$	Number of interpolation points for base energy
E_Tp<int>_X	—	—	K	—	Temperature of <int>-th interpolation point
E_Tp<int>_Y	—	—	eV	—	Energy of <int>-th interpolation point

## Chapter 18: Phase and State Transitions

### Interaction of Multistate Configurations With Transport

Alternatively, the base energy can be described as a piecewise linear (pwl) function of the temperature: With `E_nb_Tpairs`, you specify the number of interpolation points. For the `<int>`-th interpolation point ( $0 \leq <\text{int}> < \text{E_nb_Tpairs}$ ), you must specify with `E_Tp<int>_X` the temperature and with `E_Tp<int>_Y`, the corresponding energy. The pwl specification is used if `E_nb_Tpairs` is greater than zero.

Some transition parameters are inherited from the global specification (see [Table 106 on page 592](#)). They can be overwritten for specific transitions by using the appropriate parameter prefix. [Table 109](#) lists additional transition parameters.

*Table 109 Transition parameters of pmi\_ce\_msc*

Name	Symbol	Default	Unit	Range	Description
CB_nb_particles	$N_n$	0	1	integer	Particle number of reservoir CB
E_T	$E_{T, \text{user}}$	0.	eV	real	Trap energy
Eact	$E_{\text{act}}$	0.	eV	real	Activation energy
formula	—	—	—	<a href="#">Table 104</a>	Model selection
r0	$r_0$	1.	Hz	$>0.$	Maximal frequency
s0	—	—	—		Parameter index of ‘to’ state
s1	—	—	—		Parameter index of ‘from’ state
sigma	$\sigma$	$1 \times 10^{-15}$	cm <sup>2</sup>	$>0$	Cross section
VB_nb_particles	$N_p$	0	1	integer	Particle number of reservoir VB
vth	$v_{\text{th}}$	$1 \times 10^7$	cm/s	$>0$	Thermal velocity

---

## Interaction of Multistate Configurations With Transport

Multistate configurations (MSCs) have a direct impact on the transport through their charge density and their recombination rates with selected reservoirs. Furthermore, several physical models can be made explicitly dependent on the occupation probabilities of a specific MSC by using the PMI.

## Apparent Band-Edge Shift

The keywords `eBandEdgeShift`, `hBandEdgeShift`, or `BandEdgeShift` in an `MSConfig` section switch on band-edge shift models for the conduction band, the valence band, or both, respectively. The model itself is either the MSC-dependent `pmi_msc_abes` model (see [The pmi\\_msc\\_abes Model on page 595](#)) or a user-defined `PMI_MSC_ApparentBandEdgeShift` model as described in [Multistate Configuration-Dependent Apparent Band-Edge Shift on page 1323](#).

The apparent band-edge shifts become visible only if the density gradient transport model is used. To avoid the implicit use of the density gradient quantum correction model, the (electron and hole) `gamma` parameters in the `QuantumPotentialParameters` parameter set must be set to zero in the parameter file.

A typical simulation is:

```
Physics {
    MSConfigs (
        MSConfig ( ...
            eBandEdgeShift ( "pmi_abes" 0 )
            hBandEdgeShift ( "pmi_abes" 1 )
        )
    )
    eQuantumPotential
    hQuantumPotential
}
Solve {
    Coupled { Poisson Electron Hole Temperature eQuantumPotential
              hQuantumPotential }
    Transient ( ... ) {
        Coupled { Poisson Electron Hole Temperature eQuantumPotential
                  hQuantumPotential }
    }
}
```

Here, the user PMI model `pmi_abes` has been used for both the conduction and valence bands.

## The `pmi_msc_abes` Model

The `pmi_msc_abes` model depends on the lattice temperature  $T$  and, if an MSC is specified, on the state occupation probabilities  $s_i$ , and it reads as:

$$\Lambda = \sum_i \Lambda_i(T) s_i \quad (588)$$

where the sum is taken over all MSC states, and  $\Lambda_i(T)$  is the apparent band-edge shift (ABES) of state  $i$ .

## Chapter 18: Phase and State Transitions

### Interaction of Multistate Configurations With Transport

The model is activated by using (here, for electrons, a MSC named "m0", and a model index 5) as described above:

```
eBandEdgeShift ( "pmi_msc_abes" 5 )
```

The model uses a two-level or three-level hierarchy (depending on `use_mi_pars`) to determine the state parameters, namely, the global, the model index, and the state parameters. [Table 110](#) lists the global parameters.

*Table 110 Global, model-string, and state parameters of pmi\_msc\_abes*

Name	Symbol	Default	Unit	Range	Description
plot	—	0	—	{0,1}	Plot parameter to screen
use_mi_pars	—	0	—	{0,1}	Use model index for parameter set
lambda	$\Lambda$	0.	eV	real	Constant value
lambda_nb_Tpairs	—	0	—	$\geq 0$	Number of interpolation points
lambda_Tp<int>_X	—	—	K	$> 0$ .	Temperature at <int>-th interpolation point
lambda_Tp<int>_Y	—	—	eV	real	Value at <int>-th interpolation point

The names of model index parameters are prefixed with:

`mi<model_index>_`

The state parameters have one of the prefixes depending on the value of `use_mi_pars`:

`mi<model_index>_<state_name>_`  
`<state_name>_`

The `lambda_` parameters enable either a constant or pw1 apparent band-edge shift for each state, which is fully analogous to the specification described in [The pmi\\_msc\\_heatcapacity Model on page 1018](#).

---

## Thermal Conductivity, Heat Capacity, and Mobility

The following models allow the dependency on MSC occupation probabilities:

- [The pmi\\_msc\\_mobility Model on page 393](#)
- [The pmi\\_msc\\_heatcapacity Model on page 1018](#)
- [The pmi\\_msc\\_thermalconductivity Model on page 1031](#)

## Chapter 18: Phase and State Transitions

### Manipulating MSCs During Solve

Descriptions of PMIs for MSC-dependent thermal conductivity, heat capacity, and mobility can be found in [Multistate Configuration–Dependent Thermal Conductivity on page 1332](#), [Multistate Configuration–Dependent Heat Capacity on page 1329](#), and [Multistate Configuration–Dependent Bulk Mobility on page 1327](#), respectively.

---

## Manipulating MSCs During Solve

The MSC dynamic is, in general, solved implicitly. However, sometimes it might be useful to manipulate the computations. For example, you might want to initialize MSCs with nonsteady-state solutions or to freeze the dynamic of MSCs because time constants are very large compared to electronic and thermal effects. Furthermore, it might be of interest to disable specific MSC transitions for certain applications (for example, in phase-change memory applications to freeze phases but to allow electronic transitions). The available mechanisms are described here.

---

### Explicit State Occupations

You can set explicitly the state occupations of MSCs to a spatial-independent and solution-independent value, and freeze and unfreeze the dynamic of the MSCs during Solve by using `MSConfigs` in a `Set` command (see [Table 346 on page 1729](#) for a summary of options).

To set the state occupations of an MSC, you use the `MSConfig` command within `MSConfigs`, identify the MSC by using its name (and, for mixed-mode simulations, the device name using `Device`), and specify the occupancies for the involved states using `State`.

The state occupations are then set collectively and normalized implicitly, while unspecified states default to zero occupation. The `Frozen` option of `MSConfigs` applies to all existing MSCs and freezes the state occupations at the set or actual values. To unfreeze the dynamics of the MSCs again, the `-Frozen` option is used.

The following example illustrates the syntax for single-device simulations:

```
Physics {
    MSConfigs ( ...
        MSConfig (Name="msc1" State (Name="s0") State (Name="s1") ... )
    )
}
Solve { ...
    Set (
        MSConfigs (
            MSConfig ( Name="msc1"
                State (Name="s0" Value=0.1) State (Name="s1" Value=0.4) )
            Frozen
                                # freeze the MSC dynamic
        )
    )
}
```

## Chapter 18: Phase and State Transitions

Example: Two-State Phase-Change Memory Model

```
...  
Set ( MSConfigs ( -Frozen ) )           # unfreeze the MSC dynamic  
}
```

Here, the occupancies of states  $s_0$  and  $s_1$  of the MSC `msc1` are set to 0.2 and 0.8, respectively, due to the implicit normalization; all other states are unoccupied.

**Note:**

Freezing and unfreezing MSCs implies freezing and unfreezing traps, respectively (see the `Solve-Set-Traps` command in [Explicit Trap Occupation on page 550](#)).

---

## Manipulating Transition Dynamics

You can manipulate the transition dynamic by accessing prefactors for individual transition forward (capture) and backward (emission) reaction rates. This means that the modified rate  $c = \kappa c$  ( $\kappa$  is the prefactor, and  $c$  is the true reaction rate) is used in the dynamic equation. By setting selected transition prefactors to zero, you can decouple states into independent groups. Especially, if you decouple a certain state from all others, the state is frozen, that is, it retains its occupation in the subsequent analysis. With:

```
Set( (Device="d1" MSConfig="m7" Transition="t3" CPreFactor=0.  
      EPreFactor=0.) )
```

both reaction rates of the specified transition are set to zero (`PreFactor` can be used for common settings of both reaction rates). Omitting one of the specifying string keywords `Device`, `MSConfig`, and `Transition` applies the setting to all corresponding objects in the `Device-MSConfig-Transition` hierarchy.

**Note:**

The prefactors affect only subsequent computations, but they do not change the physical parameters of the MSC.

---

## Example: Two-State Phase-Change Memory Model

Phase-change memory (PCM) devices store a bit as the phase (crystalline or amorphous) of a (chalcogenide) material. Reading information takes advantage of the different conductances of the phases. Storing information requires switching the phases. To switch to the amorphous phase, a high current is passed through the material, heating up the device above the melting point. When switching off the current, the molten material cools so rapidly that it cannot crystallize, but remains in a metastable amorphous phase. To switch to the crystalline phase, the material is reheated more gently. The material remains solid, but the temperature is sufficient that the phase can relax to the equilibrium crystalline phase.

## Chapter 18: Phase and State Transitions

### References

The PCM device is modeled by a two-state model representing the crystalline and amorphous phase using an MSC with two uncharged states. The transition is modeled by the Arrhenius law formula ( $\text{Formula}=0$ ) of the `pmi_ce_msc` transition model.

The model has four parameters:

- The base energies and the degeneracy factors of the states determine the equilibrium.
- The activation energy  $E_{\text{act}}$  and  $r_0$  of the transition determine the velocity of the transition.

A short interpretation is given:

Let  $s = s_c$  denote the degree of crystallization of the material and  $s_a = 1 - s$ , the amorphization rate. Assuming an energy difference  $\delta E > 0$  between the amorphous and crystalline phases, and that the amorphous phase has  $g > 1$  times more microscopic realizations than the crystalline state, in equilibrium  $s = 1/[g \exp(-\delta E/kT) + 1]$ , that is, the amorphous state is preferred at higher temperatures. The critical temperature where the equilibrium occupation rates  $s_c$  and  $s_a$  are equal is  $T_{\text{crit}} = \delta E/k \ln(g)$ .

The dynamic is given by crystallization and amorphization rates  $r_c = r_0 \exp(-E_{\text{act}}/kT)$  and  $r_a = r_0 g \exp(-(E_{\text{act}} + \delta E)/kT)$ , respectively, assuming a simple Arrhenius law for the crystallization rate. Here,  $r_0$  is the maximal crystallization rate, and  $E_{\text{act}}$  is the activation energy.

---

## References

- [1] C. Peng, L. Cheng, and M. Mansuripur, "Experimental and theoretical investigations of laser-induced crystallization and amorphization in phase-change optical recording media," *Journal of Applied Physics*, vol. 82, no. 9, pp. 4183–4191, 1997.
- [2] S. Senkader and C. D. Wright, "Models for phase-change of  $\text{Ge}_2\text{Sb}_2\text{Te}_5$  in optical and electrical memory devices," *Journal of Applied Physics*, vol. 95, no. 2, pp. 504–511, 2004.
- [3] D.-H. Kim *et al.*, "Three-dimensional simulation model of switching dynamics in phase change random access memory cells," *Journal of Applied Physics*, vol. 101, p. 064512, March 2007.

## Degradation

---

*This chapter discusses the degradation models used in Sentaurus Device.*

### Overview of Degradation Models

A necessary part of predicting CMOS reliability is the simulation of the time dependence of interface trap generation. To cover as wide a range as possible, this simulation should accurately reflect the physics of the interface trap formation process. Although the mechanisms of interface trap generation are not completely understood, it is generally accepted that they involve silicon-hydrogen (Si-H) bond breakage and subsequent hydrogen transport, and various physical models have been proposed in the literature.

Sentaurus Device provides the following degradation models that account for time-dependent trap generation:

- The trap degradation model is a simple model that captures the reaction-diffusion theory with hydrogen atom transport in the gate oxide (see [Trap Degradation Model on page 601](#)).
- The fluence model explicitly expresses the increase in trap concentration as a function of the number of carriers that have been captured and emitted by a trap (see [Fluence Model on page 611](#)).
- The multistate configuration (MSC)–hydrogen transport degradation model accounts for 3D transport of hydrogen atoms, ions, and molecules in all regions. Complex reaction dynamics related to Si-H bond breakage at the silicon–oxide interface are described with the MSC framework (see [MSC–Hydrogen Transport Degradation Model on page 611](#)).
- The two-stage negative bias temperature instability (NBTI) degradation model, proposed by Grasser *et al.* [1] and Goes *et al.* [2], is related to the creation of  $E'$  centers and  $P_b$  centers (oxide and interface dangling bonds) (see [Two-Stage NBTI Degradation Model on page 621](#)).
- The extended nonradiative multiphonon model [3] is a four-state model that includes metastable states and accounts for the behavior of oxide defects related to NBTI (see [Extended Nonradiative Multiphonon Model on page 625](#)).

## Chapter 19: Degradation

### Trap Degradation Model

- The hot-carrier stress degradation model is a general degradation model suitable for MOS-based devices. It includes mechanisms for hot-carrier stress degradation and field-enhanced thermal degradation (see [Hot-Carrier Stress Degradation Model on page 631](#)).
- The activated barrier double well thermionic model can be used to describe the contribution to the overall threshold voltage shift during and after NBTI or positive bias temperature instability (PBTI) due to hole or electron trapping and detrapping from process-dependent preexisting traps in the gate insulator (see [Activated Barrier Double Well Thermionic Model on page 638](#)).
- The transient trap occupancy model can be used to compute the charge occupancy of generated interface traps contributing to the overall threshold voltage shift for bias temperature instability calculations (see [Transient Trap Occupancy Model on page 641](#)).

---

## Trap Degradation Model

Disorder-induced variations among the Si–H activation energies at the passivated Si–SiO<sub>2</sub> interface have been shown [4] to be a plausible source of the sublinear time dependence of this trap generation process. Diffusion of hydrogen from the passivated interface was used to explain some time dependencies [5]. In addition, this could be due to a Si–H density–dependent activation energy [6], which might be due to the effects of Si–H breaking on the electrical and chemical potential of hydrogens at the interface. Furthermore, the field dependence of the activation energy due to the Poole–Frenkel effect can be considered, so that all these factors lead to enhanced trap formation kinetics.

---

## Trap Formation Kinetics

The main assumption about trap formation is that initially dangling silicon bonds at the Si–SiO<sub>2</sub> interface were passivated by hydrogen (H) or deuterium (D) [7], and degradation is a depassivation process where hot carrier interactions with Si–H/D bonds or other mechanisms are responsible for this. The equations of the model are solved self-consistently with all transport equations.

## Power Law and Kinetic Equation

The experimental data for the kinetics of interface trap formation [8] shows that the time dependence of trap generation can be described by a simple power law:

$$N_{it} - N_{it}^0 = N_{hb}^0 / (1 + (vt)^{-\alpha}),$$
where  $N_{it}$  is the concentration of interface traps, and  $N_{hb}^0$  and  $N_{it}^0$  are the initial concentrations of Si–H bonds (or the concentration of hydrogen on Si bonds) and interface traps, respectively.

## Chapter 19: Degradation

### Trap Degradation Model

Assuming  $N = N_{\text{hb}}^0 + N_{\text{it}}^0$  total Si bonds at the interface, the remaining number of Si-H bonds at the interface after stress is  $N_{\text{hb}} = N - N_{\text{it}}$  and follows the power law:

$$N_{\text{hb}} = \frac{N_{\text{hb}}^0}{1 + (vt)^\alpha} \quad (589)$$

Based on experimental observations, the power  $\alpha$  is stress dependent and varies between 0 and 1.

From first-order kinetics [4], it is expected that the Si-H concentration during stress obeys:

$$\frac{dN_{\text{hb}}}{dt} = -vN_{\text{hb}} \quad (590)$$

where  $v$  is a reaction constant that can be described by  $v \propto v_A \exp(-\epsilon_A/kT)$  in the Arrhenius approximation,  $\epsilon_A$  is the Si-H activation energy, and  $T$  is the Si-H temperature. The exponential kinetics given by this equation ( $N_{\text{hb}} = N_{\text{hb}}^0 \exp(-vt)$ ) do not fully describe the experimental data because a constant activation energy will behave like the power law in Equation 589, but with power  $\alpha \approx 1$ .

## Si-H Density–Dependent Activation Energy

This section describes an activation energy parameterization to capture the sublinear power law for the time dependence of interface trap generation. There is evidence that the hydrogen atoms, when removed from the silicon, remain negatively charged [9]. If this correct, the hydrogen can be expected to remain in the vicinity of the interface and will affect the breaking of additional silicon-hydrogen bonds by changing the electrical potential.

The concentration of released hydrogen is equal to  $N - N_{\text{hb}}$ , so the activation energy dependence (assuming the activation energy changes logarithmically with the breaking of Si-H) can be expressed as:

$$\epsilon_A = \epsilon_A^0 + (1 + \beta)kT \ln \frac{N - N_{\text{hb}}}{N - N_{\text{hb}}^0} \quad (591)$$

where the last term represents the Si-H density–dependent change with a prefactor  $1 + \beta$ . Note that  $(N - N_{\text{hb}})/(N - N_{\text{hb}}^0)$  is the fraction of traps generated to the total initial traps, and this gives the form of the chemical potential of Si-H bonds with the prefactor  $1 + \beta$ .

The numeric solution of the kinetic equation with the varying activation energy above clearly shows that such a Si-H density–dependent activation energy gives a power law, and the power  $\alpha$  is a function of the prefactor  $1 + \beta$ . From the available experimental data of interface trap generation, it was noted that for negative gate biases  $\alpha > 0.5$ , but for positive ones  $\alpha < 0.5$ . It is interesting that the solution of the above kinetic equation gives  $\alpha \approx 0.5$  in the equilibrium case where a unity prefactor is used. In nonequilibrium, a polarity-dependent modification of the prefactor (field stretched and pressed Si-H bonds) is possible.

## Diffusion of Hydrogen in Oxide

Another model that can interpret negative bias temperature instability (NBTI) phenomena and different experimental slopes in degradation kinetics is the R-D model [10].

This model considers hydrogen in oxide, which diffuses from the silicon–oxide interface, but the part of the hydrogen that remains at the interface controls the degradation kinetics.

The diffusion of hydrogen in oxide can be expressed as follows:

$$\begin{aligned} D \frac{dN_H}{dx} &= \frac{dN_{hb}}{dt} & x = 0 \\ \frac{dN_H}{dt} &= D \frac{d^2N_H}{dx^2} & 0 < x < x_p \\ D \frac{dN_H}{dx} &= -k_p(N_H - N_H^0) & x = x_p \end{aligned} \quad (592)$$

where  $N_H$  is a concentration of hydrogen in oxide,  $D = D_0 \exp(-\epsilon_H/kT)$  is its diffusion coefficient,  $x = 0$  is a coordinate of the silicon–oxide interface,  $x = x_p$  is the coordinate of the oxide–polysilicon gate interface (which is equal to the oxide thickness),  $k_p$  is the surface recombination velocity at the oxide–polysilicon gate interface, and  $N_H^0$  is an equilibrium (initial) concentration of hydrogen in the oxide.

**Note:**

The model solves the one-dimensional diffusion equation (Equation 592) with  $x_p$  as the effective oxide thickness in a critical part of the interface where the degradation occurs. To solve Equation 592 numerically, a uniform mesh is used with a number of nodes  $N_{ox}$ . Typically, a 15-node mesh is considered to be sufficiently practical.  $N_{ox}$  should be defined together with other model parameters (see [Using the Trap Degradation Model on page 606](#)).

## Model Equations and Syntax

To use the trap degradation model, specify the `Degradation` keyword as part of a `Traps` specification in the command file. This invokes the general kinetic equation (left column of [Equation 593](#)) with an added passivation term. To use the power law form of the model (right column of [Equation 593](#)), specify `Degradation(PowerLaw)`:

Kinetic	Power Law
$\frac{dN_{hb}}{dt} = -vN_{hb} + \gamma(N - N_{hb})$	$N_{hb} = \frac{N_{hb}^0}{1 + (vt)^\alpha}$
$\gamma = \gamma_0[N_H/N_H^0 + \Omega(N_{hb}^0 - N_{hb})]$	$\alpha = 0.5 + \beta$
$\gamma_0 = \frac{N_{hb}^0}{N - N_{hb}^0} v_0$	

(593)

## Chapter 19: Degradation

### Trap Degradation Model

In [Equation 593](#),  $\gamma_0$  is the passivation constant, which is computed automatically by default to provide the equilibrium, but it can also be specified directly in the command file.  $\Omega$  is the passivation volume (by default, it is equal to zero) and represents a simple model for the retrapping of depassivated hydrogen by dangling silicon bonds. Depassivated hydrogen increases the average hydrogen concentration  $N_H$  near traps, which is computed from the R-D model (see [Diffusion of Hydrogen in Oxide on page 603](#)). If the R-D model is not activated, then  $N_H/N_H^0 = 1$ .

The parameterized system of equations for the reaction constant  $v$ , based on the trap formation model (see [Trap Formation Kinetics on page 601](#) and [6]), can be expressed as:

$$\begin{aligned} v &= v_0 \exp \frac{\epsilon_A^0}{kT_0} - \frac{\epsilon_A^0 + \Delta\epsilon_A}{\epsilon_T} k_{HC} k_{Tun} k_{SHE} \\ \epsilon_T &= kT + \delta_{//}|F_{//}|^{\rho_{//}} \\ \Delta\epsilon_A &= -\delta_{\perp}|F_{\perp}|^{\rho_{\perp}} + (1 + \beta)\epsilon_T \ln \frac{N - N_{hb}^0}{N - N_{hb}} \\ \beta &= \beta_0 + \beta_{\perp}F_{\perp} + \beta_{//}F_{//} \end{aligned} \quad (594)$$

where:

- $v_0$  (given in the command file) is the reaction (depassivation) constant at the passivation equilibrium (for the passivation temperature  $T_0$  and for no changes in the activation energy  $\Delta\epsilon_A = 0$ ).
- $F_{\perp}$ ,  $F_{//}$  are the perpendicular and parallel components of the electric field  $F$  to the interface where traps are located. The perpendicular electric field  $F_{\perp}$  has a positive sign if the space charge at the interface is positive.
- $\epsilon_T$  is the energy of hydrogen on Si-H bonds and is equal to  $kT$  plus some possible gain from hot carriers represented as the additional term that is dependent on the parallel component of the electric field  $\delta_{//}|F_{//}|^{\rho_{//}}$ .
- $\Delta\epsilon_A$  is a change of the activation energy because of stretched Si-H bonds [11] by the electric field (first term) and due to a change of the chemical potential (second term) [6]. Effectively, the influence of the chemical potential also can be different in the presence of the electric field, and the coefficient  $\beta$  represents this.
- The coefficients  $\delta_{\perp}$ ,  $\rho_{\perp}$ ,  $\delta_{//}$ ,  $\rho_{//}$  and  $\beta_0$ ,  $\beta_{\perp}$ ,  $\beta_{//}$  are field enhancement parameters for the model.

## Reaction Enhancement Factors

In [Equation 594](#),  $k_{HC}$  and  $k_{Tun}$  are reaction enhancement factors due to hot-carrier and tunneling current, respectively. They are given by:

$$k_{HC} = 1 + \delta_{HC} \left| \frac{I_{HC}}{I_0} \right|^{\rho_{HC}} \quad (595)$$

$$k_{Tun} = 1 + \delta_{Tun} \delta_{FN} \left| \frac{I_{FN}}{I_0} \right|^{\rho_{FN}} + \delta_{DTe} \left| \frac{I_{DTe}}{I_0} \right|^{\rho_{DTe}} + \delta_{DTh} \left| \frac{I_{DTh}}{I_0} \right|^{\rho_{DTh}} + \delta_{BTe} \left| \frac{I_{BTe}}{I_0} \right|^{\rho_{BTe}} + \delta_{BTh} \left| \frac{I_{BTh}}{I_0} \right|^{\rho_{BTh}} \quad (596)$$

where  $I_{HC}$  is the local hot-carrier current density (see [Chapter 25 on page 844](#)), and  $I_{FN}$ ,  $I_{DTe}$ ,  $I_{DTh}$ ,  $I_{BTe}$ , and  $I_{BTh}$  are the tunneling current densities from Fowler–Nordheim tunneling, electron and hole direct tunneling, and electron and hole nonlocal barrier tunneling, respectively (see [Chapter 24 on page 819](#)). In these equations,  $I_0 = 1 \text{ A/cm}^2$ .

When the electron energy distribution is available by specifying the `eSHEDistribution` keyword in the `Physics` section (see [Using Spherical Harmonics Expansion Method on page 857](#)), you can include the following additional spherical harmonics expansion (SHE) distribution enhancement factor:

$$k_{SHE} = 1 + \delta_{SHE} \frac{q g_v}{2} \sum_{\epsilon_{th}}^{\infty} \min \left[ \exp \left( \frac{\epsilon - \epsilon_a + \delta_{\perp} |F_{\perp}/F_0|^{p_{\perp}}}{kT} \right), 1 \right] g(\epsilon) f(\epsilon) v(\epsilon) d\epsilon^{\rho_{SHE}} \quad (597)$$

where:

- $\delta_{SHE}$  is a prefactor.
- $\epsilon_{th}$  is a threshold energy.
- $\epsilon_a$  is an activation energy.
- $\delta_{\perp}$  is a normal field-induced activation energy-lowering factor.
- $p_{\perp}$  is a normal field-induced activation energy-lowering exponent.
- $F_0 = 1 \text{ V/cm}$ .
- $g_v$  is the valley degeneracy.
- $g$  is the density-of-states.
- $f$  is the electron energy distribution.
- $v$  is the magnitude of the electron velocity.
- $\rho_{SHE}$  is an exponent for the SHE current.

The SHE distribution enhancement factor can be specified by the `eSHEDistribution` keyword in the `Traps` statement. Similarly, the `hSHEDistribution` keyword specifies the enhancement factor of the hole energy distribution.

## Chapter 19: Degradation

### Trap Degradation Model

The following example specifies the SHE distribution enhancement factor with  $\delta_{\text{SHE}} = 100 \text{ cm}^2/\text{A}$ ,  $\varepsilon_{\text{th}} = 2 \text{ eV}$ ,  $\varepsilon_a = 2.1 \text{ eV}$ ,  $\delta_{\perp} = 10^{-7} \text{ eV}$ ,  $\rho_{\perp} = 1$ , and  $\rho_{\text{SHE}} = 1$ :

```
Physics(MaterialInterface="Silicon/Oxide") {
    Traps(... {
        Degradation
        # (delta_SHE E_th E_a delta_p rho_p rho_SHE)
        eSHEDistribution=(1.0e2 2 2.1 1.0e-7 1 1)
        ...
    })
}
```

## Using the Trap Degradation Model

The Degradation model can be activated for any trap level or distribution (see [Chapter 17 on page 543](#)). The keywords related to the Degradation model are described in [Table 331 on page 1701](#), along with all other trap-related options, and their syntax is shown here:

```
Physics ( [ RegionInterface | MaterialInterface = "<name1>/<name2>" | ]
           [ Material | Region = "<name>" ] ) {
    Traps (
        <trap_specifications>
        Degradation [(PowerLaw)]
        Conc=<Nit0>
        BondConc=<N> [BondConcSFactor=[ "<dataset_name>" |
                           "<pmi_model_name>" ]]
        ActEnergy=<εA0>
        DePasCoef=<ν0>
        [CritConc=<Ncrit>]
        [FieldEnhan=(<δ//> <ρ//> <δ⊥> <ρ⊥>)]
        [PowerEnhan=(<β0> <β//> <β⊥>)]
        [CurrentEnhan=(<δTun> <ρTun> <δHC> <ρHC>)]
        [FowlerNordheimEnhan=(<δFN> <ρFN>)]
        [DirectTunnelingEnhan=(<δDTe> <ρDTe> <δDTh> <ρDTh>)]
        [BarrierTunnelingEnhan=(<δBTe> <ρBTe> <δBTh> <ρBTh>)]
        [eSHEDistribution=(<δSHE> <εth> <εa> <δ⊥> <ρ⊥> <ρSHE>)]
        [hSHEDistribution=(<δSHE> <εth> <εa> <δ⊥> <ρ⊥> <ρSHE>)]
        [DiffusionEnhan=(<xp> <D0> <εH> <kp> <NH0> <Nox>)]
        [PasCoef=<γ0>]
        [PasTemp=<T0>]
        [PasVolume=<Ω>]
    )
    ...
}
```

## Chapter 19: Degradation

### Trap Degradation Model

The total silicon-bond concentration, which represents the maximum concentration of traps that can be generated as the result of degradation, is specified with `BondConc`. It is possible to have a spatially dependent bond concentration by using the parameter `BondConcSFactor` to specify a dataset or a PMI space factor model (see [Space Factor on page 1479](#)) that describes the spatial dependency. If `BondConcSFactor` is specified without `BondConc`, then the values obtained from the dataset or PMI are used directly. If both `BondConcSFactor` and `BondConc` are specified, then the `BondConcSFactor` values are normalized by the largest `BondConcSFactor` value and are then multiplied by `BondConc`.

---

## Device Lifetime and Simulation

The following example illustrates the use of the `Degradation` model:

```
Physics(MaterialInterface="Silicon/Oxide"){
    Traps(Conc=1e8 EnergyMid=0 Acceptor #FixedCharge
        Degradation #(PowerLaw)
        ActEnergy=2 BondConc=1e12
        DePasCoeff=8e-10
        FieldEnhan=(0 1 1.95e-3 0.33)
        CurrentEnhan=(0 1 6e+5 1)
        PowerEnhan=(0 0 -1e-7)
    )
    GateCurrent(GateName="gate" Lucky(CarrierTemperatureDrive) Fowler)
}
```

For this input, the initially specified trap concentration is  $10^8 \text{ cm}^{-2}$  and, in the process of degradation, it can be increased up to  $10^{12} \text{ cm}^{-2}$ . The activation energy of hydrogen on Si-H bonds is 2 eV and the depassivation constant at the equilibrium is equal to  $8 \times 10^{-10} \text{ s}^{-1}$ . The degradation simulation can be separated into two parts:

- Simulation of extremely stressed devices with existing experimental data and fitting to the data by modification of the field and current-dependent parameters.
- Simulation of normal-operating devices to predict device reliability (lifetime).

For the first part of the degradation simulation, the typical `Solve` section can be:

```
Solve {
    NewCurrentPrefix="tmp"
    coupled (iterations=100) { Poisson }
    coupled { poisson electron hole }
    Quasistationary( InitialStep=0.1 MaxStep=0.2 MinStep=0.0001
        increment=1.5 Goal{name="gate" voltage=-10} )
    { coupled { poisson electron hole } }
    NewCurrentPrefix=""
    coupled { poisson electron hole }
    transient( InitialTime=0 Finaltime = 100000
        increment=2 InitialStep=0.1 MaxStep=100000 ){
        coupled{ poisson electron hole }
```

## Chapter 19: Degradation

### Trap Degradation Model

```
    }  
}
```

The first Quasistationary ramps the device to stress conditions (in this particular case, to high negative gate voltage), the second transient simulates the degradation kinetics (up to  $10^5$  s, which is a typical time for stress experimental data).

#### Note:

The hot carrier currents are postprocessed values and, therefore, InitialStep should not be large.

To monitor the trap formation kinetics in transient, you can use Plot and CurrentPlot statements to output TotalTrapConcentration and OneOverDegradationTime. For example:

```
CurrentPlot{  
    eDensity(359) Potential(359)  
    OneOverDegradationTime(359)  
    TotalTrapConcentration(359)  
}
```

where a vertex number is specified to have a plot of the fields at some location on the interface. As a result, the behavior of these values versus time can be seen in the plot file of Sentaurus Device.

The prediction of the device lifetime can be performed in two different ways:

- Direct simulation of a normal-operating device in transient for a long time (for example, 30 years).
- Extrapolation of the degradation of a stressed device by computation of the ratio between depassivation constants for stressed and unstressed conditions.

The important value here is the critical trap concentration  $N_{\text{crit}}$ , which defines an edge between a properly working and improperly working device. Using  $N_{\text{crit}}$ , the device lifetime  $\tau_D$  is defined as follows (according to different prediction ways):

1. In transient, direct computation of time  $t = \tau_D$  gives the trap concentration equal to  $N_{\text{crit}}$ .
2. In Quasistationary, if the previously finished transient statement computes the device lifetime  $\tau_D^{\text{stress}}$  and the depassivation constant  $v^{\text{stress}}$  at stress conditions, then  $\tau_D = (v^{\text{stress}}/v)\tau_D$ .

So, the plotted value of OneOverDegradationTime is equal to  $1/\tau_D$  for one trap level and the sum  $1/\tau_D^i$  if several trap levels are defined for the degradation. It is computed for each vertex where the degradation model is applied and can be considered as the lifetime of local device area.

## Chapter 19: Degradation

### Trap Degradation Model

For the second approach to device lifetime computation, the following `Solve` statement can be used:

```
Solve {
    NewCurrentPrefix="tmp"
    coupled (iterations=100) { Poisson }
    coupled { poisson electron hole }

    Quasistationary( InitialStep=0.1 MaxStep=0.2 Minstep=0.0001
                      increment=1.5 Goal{name="gate" voltage=-10} )
                      { coupled { poisson electron hole } }

    NewCurrentPrefix=""
    coupled { poisson electron hole }
    transient( InitialTime=0 Finaltime = 100000
               increment=2 InitialStep=0.1 MaxStep=100000 ){
        coupled{ poisson electron hole }
    }

    set(Trapfilling=-Degradation)

    coupled { poisson electron hole }
    Quasistationary( InitialStep=0.1 MaxStep=0.2 Minstep=0.0001
                      increment=1.5
                      Goal{name="gate" voltage=1.5} )
                      { coupled { poisson electron hole } }
    Quasistationary( InitialStep=0.1 MaxStep=0.2 Minstep=0.0001
                      increment=1.5
                      Goal{name="drain" voltage=3} )
                      { coupled { poisson electron hole } }
}
```

The statement `set(Trapfilling=-Degradation)` returns the trap concentrations to their unstressed values (it is not necessary to include, but it might be interesting to check the influence). The first `Quasistationary` statement after the `set` command returns the normal-operating voltage on the gate and, in the second, you can plot the dependence of  $1/\tau_D$  on applied drain voltage. The last dependence could be useful to predict an upper limit of operating voltages where the device will work for a specified time.

---

## Degradation in Insulators

Although the `Degradation` model was designed for trap generation that occurs at semiconductor-insulator interfaces, the model can be used to generate traps at insulator-insulator interfaces or in bulk-insulator regions.

For insulator degradation, Sentaurus Device sets  $F_{\perp} = F$  and  $F_{\parallel} = 0$ , and the enhancement factors  $k_{HC}$ ,  $k_{Tun}$ , and  $k_{SHE}$  are taken from values computed at the semiconductor-insulator interface.

## Chapter 19: Degradation

### Trap Degradation Model

#### Note:

To allow traps that are created in insulator regions to be filled with carriers, it is necessary to construct a nonlocal mesh to connect the semiconductor–insulator interface to the regions where traps are generated and to invoke nonlocal tunneling models (see [Nonlocal Tunneling for Traps on page 576](#)).

To use this feature:

1. Create a nonlocal mesh that connects the semiconductor interface with the regions where traps are generated. For example:

```
Math {
    NonLocal "NLM_silicon_oxide" (
        MaterialInterface = "Silicon/Oxide"
        Length = 4.1e-7      # Use a length that will reach the traps
    )
}
```

2. Specify nonlocal tunneling and `Traps(Degradation ...)` in the `Physics` section associated with the insulator–insulator interface or the bulk-insulator region where degradation will be used. Degradation parameters must be specified and adjusted as necessary. For example:

```
Physics (RegionInterface="oxide1/oxide2") {
    Traps (
        eBarrierTunneling(Nonlocal="NLM_silicon_oxide")
        hBarrierTunneling(Nonlocal="NLM_silicon_oxide")
        TrapVolume=1e-6
        HuangRhys=70
        PhononEnergy=0.02
        #
        Degradation
        Conc=1e4 EnergyMid=0 Donor
        ActEnergy=2
        BondConc=1e14
        DePasCoeff=1e-11
        PasTemp=300
        FieldEnhan=(0 1 3.9e-3 0.33)
        CurrentEnhan=(0 1 1 1)
        PowerEnhan=(0 0 -1e-7)
        DiffusionEnhan=(2e-7 1e-13 0.05 1.0 5e10 15)
        PasVolume=5e-10
    )
}
```

3. In the parameter file, specify a nonzero tunneling mass for all regions or materials where tunneling can occur.

For example:

```
Material = "Silicon" {
    BarrierTunneling "NLM_silicon_oxide" { mt = 1, 1 }
}
Material = "Oxide" {
    BarrierTunneling "NLM_silicon_oxide" { mt = 1, 1 }
}
```

---

## Fluence Model

In the fluence model, degradation is described as an increase in trap concentration as a function of the number of carriers captured and emitted by a particular trap. In this model, the trap concentration at time  $t$ ,  $N_{\text{nb}}(t)$ , is given by the initial trap concentration  $N_{\text{hb}}^0$  as:

$$N_{\text{hb}}(t) = N_{\text{hb}}^0 [(f_{\text{cb}}N_{\text{cb}}(t) + f_{\text{vb}}N_{\text{vb}}(t))^{\beta_F} + 1] \quad (598)$$

Here,  $f_{\text{cb}}$ ,  $f_{\text{vb}}$ , and  $\beta_F$  are dimensionless fit parameters. The fluences  $N_{\text{cb}}(t)$  and  $N_{\text{vb}}(t)$  are the total number of carriers that a single trap has emitted or captured from the conduction band and the valence band, respectively, from the beginning of the simulation up to time  $t$ .

**Note:**

The fluence model makes no assumption about the physical nature of the degradation process. It is merely a heuristic calibration formula. In addition, since the fluences are the sum of capture and emission processes, they grow all the time, even in equilibrium. Only far from equilibrium, you can consider the fluence as the total current that has passed through a trap.

To activate the fluence model, specify `FluenceDependence` in a `Trap` entry for a trap that should degrade. With this keyword, you specify the parameters  $\beta_F$ ,  $f_{\text{cb}}$ , and  $f_{\text{vb}}$ .

The following example specifies a midgap trap with an initial concentration of  $1\text{cm}^{-3}$ ,  $\beta_F = 0.5$ ,  $f_{\text{cb}} = 1\times 10^{-10}$ , and  $f_{\text{vb}} = 2\times 10^{-10}$ :

```
Traps((Conc=1 eNeutral EnergyMid=0 FluenceDependence=(0.5 1e-10 2e-10)))
```

---

## MSC–Hydrogen Transport Degradation Model

Bias and temperature stresses generate interface fixed charges and trapped charges near the oxide interface. These immobile charges affect the threshold voltage (see [Chapter 7 on page 229](#)) and the carrier mobility (see [Mobility Degradation Components due to Coulomb Scattering on page 419](#)). To explain these degradation phenomena, various physical models have been proposed [\[1\]](#)[\[12\]](#)[\[13\]](#)[\[14\]](#). Although the details of these models are

## Chapter 19: Degradation

### MSC–Hydrogen Transport Degradation Model

different, they commonly involve charge-trapping, silicon-hydrogen bond depassivation, and hydrogen transport.

To model charge-trapping, interactions between localized trapping centers and mobile carriers must be considered. Contrary to the standard traps in [Chapter 17 on page 543](#), the trapping centers used in the degradation model usually involve hydrogens. Therefore, interactions between the localized trapping centers and mobile hydrogens should be considered. In addition, the trapping centers might have more than two internal states depending on the structural relaxation and the presence of charges and hydrogens [1].

The multistate configurations (MSCs) introduced in [Chapter 18 on page 584](#) can be used to represent these complex trapping centers because the MSCs can handle an arbitrary number of states and their transitions involving the capture and emission of charges and hydrogens.

Finally, hydrogen transport must be considered if the degradation model involves mobile hydrogens. Hydrogen atoms, hydrogen molecules, and hydrogen ions can contribute to the hydrogen transport, and there can be chemical reactions between them [12][13][14].

Sentaurus Device provides the following capabilities to model oxide degradation:

- Transport equations for hydrogen atoms, hydrogen molecules, and hydrogen ions.
- An arbitrary number of interface and bulk reactions among mobile elements (hydrogen atoms, hydrogen molecules, hydrogen ions, electrons, and holes).
- An arbitrary number of interface and bulk reactions among mobile elements and localized states by using the multistate configurations (see [Chapter 18 on page 584](#)).

---

## Hydrogen Transport

Transport equations for hydrogen atoms ( $X_1$ ), hydrogen molecules ( $X_2$ ), and hydrogen ions ( $X_3$ ) can be written as:

$$\frac{\partial}{\partial t}[X_i] + \nabla \cdot \left[ D_i \exp -\frac{E_{di}}{kT} \frac{qK_i^Q}{kT} \vec{F}[X_i] - \nabla[X_i] - \alpha_{td}[X_i] \nabla \ln T \right] + R_{\text{net}} + r_i([X_i] - [X_i]_0) = 0 \quad (599)$$

where:

- $D_i$  is the diffusion coefficient.
- $E_{di}$  is the diffusion activation energy.
- $\alpha_{td}$  is the prefactor of the thermal diffusion term.
- $K_i^Q$  is the number of charges for element  $X_i$ .
- $R_{\text{net}}$  is the net recombination rate due to chemical reactions.

## Chapter 19: Degradation

### MSC–Hydrogen Transport Degradation Model

- $r_i$  is the explicit recombination rate.
- $[X_i]_0$  is the initial density.

By default, the initial densities of hydrogen atoms, molecules, and ions are all zero. You can load the initial density of each hydrogen element from the `PMIUserField` by using the `Set` command in the `Solve` section. For example:

```
Solve {
    ...
    Set ( HydrogenAtom = "PMIUserField0" )
    Set ( HydrogenMolecule = "PMIUserField1" )
    Set ( HydrogenIon = "PMIUserField3" )
    Set ( HydrogenSpeciesA = "PMIUserField4" )
    Set ( HydrogenSpeciesB = "PMIUserField5" )
    Set ( HydrogenSpeciesC = "PMIUserField6" )
    ...
}
```

Sentaurus Device uses default values for  $D_i \exp(-E_{di}/(kT))$  and  $\alpha_{td}$ . You can customize the computation of  $D_i \exp(-E_{di}/(kT))$  and  $\alpha_{td}$  by using PMI models (for details, see [Diffusivity on page 1398](#)). The corresponding command file can be written as:

```
Physics ( Material = "Oxide" ) {
    HydrogenDiffusion(
        HydrogenAtom (
            Diffusivity = pmi_HydrogenDiffusivity
            Alpha = pmi_HydrogenAlpha
        )
        HydrogenMolecule (
            Diffusivity = pmi_HydrogenDiffusivity
            Alpha = pmi_HydrogenAlpha
        )
        HydrogenIon (
            Diffusivity = pmi_HydrogenDiffusivity
            Alpha = pmi_HydrogenAlpha
        )
        ...
    )
}
```

## Boundary Conditions

At electrodes,  $[X_i] = [X_i]_0$  is assumed as the default boundary condition. You can specify more flexible conditions at each contact. The corresponding command file can be written as:

```
HydrogenBoundary {
    { Name = "cSiTop"
        HydrogenAtom = 1e7                      # [cm-3]
        HydrogenMolecule = reflective
    }
    { Name = "cSiBot"
```

```

    HydrogenAtom = reflective
}
...
}
```

## Reactions Between Mobile Elements

In the model, you can specify an arbitrary number of bulk and interface chemical reactions between hydrogen atoms ( $X_1$ ), hydrogen molecules ( $X_2$ ), hydrogen ions ( $X_3$ ), electrons ( $X_4$ ), and holes ( $X_5$ ). Each reaction is defined by the following reaction equation:



where the nonnegative integers  $\alpha_i$  and  $\beta_i$  are the particle numbers of element  $X_i$  to be removed and created by the forward reaction.

These coefficients must satisfy the charge and hydrogen conservation laws:

$$\sum_{i=1}^5 (\alpha_i - \beta_i) K_i^Q = 0 \quad (601)$$

$$\sum_{i=1}^5 (\alpha_i - \beta_i) K_i^H = 0 \quad (602)$$

where  $K_i^Q$  and  $K_i^H$  are the number of charges and the number of hydrogen atoms for element  $X_i$  (for  $K_i^Q$  and  $K_i^H$  of each mobile element, see [Table 103 on page 585](#)).

The forward and reverse reaction rates  $R_f$  and  $R_r$  are modeled as:

$$R_f = k_f \exp \left( \delta_f F - \frac{E_f}{kT} \right) \prod_{i=1}^5 \frac{[X_i]^{\alpha_i}}{1 / \text{cm}^3} \quad (603)$$

$$R_r = k_r \exp \left( \delta_r F - \frac{E_r}{kT} \right) \prod_{i=1}^5 \frac{[X_i]^{\beta_i}}{1 / \text{cm}^3} \quad (604)$$

where:

- $F$  is the magnitude of the electric field [V/cm].
- $k_f$  and  $k_r$  are the forward and reverse reaction coefficients, respectively ( $[\text{cm}^{-3} \text{s}^{-1}]$  for bulk reactions and  $[\text{cm}^{-2} \text{s}^{-1}]$  for interface reactions).
- $\delta_f$  and  $\delta_r$  are the forward and reverse reaction field coefficients, respectively [ $\text{cm V}^{-1}$ ].
- $E_f$  and  $E_r$  are the forward and reverse reaction activation energies, respectively [eV].

## Chapter 19: Degradation

### MSC–Hydrogen Transport Degradation Model

#### Note:

For a reaction specified at a semiconductor–insulator interface, the insulator electric field is used instead of the semiconductor electric field.

A reaction can be specified in the `Physics` section as an argument `HydrogenReaction` to the `HydrogenDiffusion` keyword. For example, consider the dimerization of two hydrogen atoms into a hydrogen molecule  $2\text{H} \leftrightarrow \text{H}_2$  in the oxide region with  $k_f = 10^{-3} \text{ cm}^{-3} \text{ s}^{-1}$  and  $k_r = 10^2 \text{ cm}^{-3} \text{ s}^{-1}$ .

The corresponding command file can be written as:

```
Physics ( Material = "Oxide" ) {
    HydrogenDiffusion(
        HydrogenReaction( #  $2\text{H} \leftrightarrow \text{H}_2$ 
            LHSCoef ( # alpha_i
                HydrogenAtom = 2
                HydrogenMolecule = 0
                HydrogenIon = 0
                Electron = 0
                Hole = 0
            )
            RHSCoef ( # beta_i
                HydrogenAtom = 0
                HydrogenMolecule = 1
                HydrogenIon = 0
                Electron = 0
                Hole = 0
            )
            ForwardReactionCoef = 1.0e-3 # k_f
            ReverseReactionCoef = 1.0e2 # k_r
            ForwardReactionEnergy = 0 # E_f
            ReverseReactionEnergy = 0 # E_r
            ForwardReactionFieldCoef = 0 # delta_f
            ReverseReactionFieldCoef = 0 # delta_r
        )
        ...
    )
}
```

The default values of all the coefficients are zero. [Table 311 on page 1683](#) lists the options available for the reaction specification.

For heterointerfaces, the keyword `Region` or `Material` allows you to specify the region or material where the reaction process enters as a generation–recombination rate. In addition, the keyword `FieldFromRegion` or `FieldFromMaterial` allows you to specify the region or material where the electric field is obtained. For example:

```
Physics ( Material = "Silicon/OxideAsSemiconductor" ) {
    HydrogenDiffusion(
        HydrogenReaction(...)
            Material = "Silicon"
```

```

        FieldFromMaterial = "OxideAsSemiconductor"
    )
}
}
```

## Reactions With Multistate Configurations

A multistate configuration (MSC) can be used to model reactions between the mobile hydrogen elements and localized hydrogen states such as silicon-hydrogen bonds at the silicon–oxide interface.

For example, consider a hydrogen depassivation model based on the hole capture process:



where Si-H, p, Si<sup>+</sup>, and H represent the silicon-hydrogen bond, hole, silicon dangling bond, and hydrogen atom, respectively. This reaction can be specified by a two-state MSC defined at the silicon–oxide interface:

```

Physics ( MaterialInterface = "Silicon/Oxide" ) {
    MSConfigs (
        MSConfig ( Name = "SiHBond" Conc=5.0e12
            State ( Name = "s1" Hydrogen=1 Charge=0 ) # Si-H bond
            State ( Name = "s2" Hydrogen=0 Charge=1 ) # Si+ dangling bond
            Transition ( Name = "t21" # Si-H + p <-> H + Si+
                To="s2" From="s1" CEModel("CEModel_Depassivation" 1)
                Reservoirs("VB"(Particles=+1) "HydrogenAtom"(Particles=-1) )
                FieldFromInsulator
            )
        )
        ...
    )
}
```

The capture and emission rates are obtained from:

$$c_{21} = c_{\text{PMI}}(n, p, T, T_n, T_p, F) \prod_{i=1}^3 \frac{[X_i]}{\text{cm}^3}^{\alpha_i} \quad (606)$$

$$e_{21} = e_{\text{PMI}}(n, p, T, T_n, T_p, F) \prod_{i=1}^3 \frac{[X_i]}{\text{cm}^3}^{\beta_i} \quad (607)$$

where  $c_{\text{PMI}}$  and  $e_{\text{PMI}}$  are the capture and emission rates specified by the trap capture and emission PMI model, respectively.

For more information about multistate configurations, see [Chapter 18 on page 584](#).

## Chapter 19: Degradation

### MSC–Hydrogen Transport Degradation Model

#### Note:

Contrary to the conventional trap capture and emission PMI model, the insulator electric field can be used in the PMI model at the semiconductor–insulator interface instead of the semiconductor electric field when the keyword `FieldFromInsulator` is set in the transition.

---

## The CEModel\_Depassivation Model

The built-in capture and emission model `CEModel_Depassivation` models the hydrogen depassivation process. In the model, the hydrogen depassivation (electron capture) rate induced by hot-electron distribution is written as:

$$c_{\text{PMI}} = 2g_v \sigma v(\varepsilon)g(\varepsilon)f(\varepsilon) \frac{\varepsilon - \varepsilon_{\text{crit}}}{kT}^p \exp\left[\gamma \frac{F}{1 \text{ V/cm}}^\rho - \frac{\Theta(W_f - q\alpha F - \chi\varepsilon)(W_f - q\alpha F - \chi\varepsilon)}{kT}\right] \quad (608)$$

where:

- $\varepsilon_{\text{min}}$  and  $\varepsilon_{\text{max}}$  are the minimum and maximum kinetic energies of the integration.
- $\sigma$  is the capture cross-section.
- $g_v$  is the valley degeneracy.
- $v(\varepsilon)$  is the magnitude of the group velocity.
- $g(\varepsilon)$  is the density-of-states.
- $f(\varepsilon)$  is the distribution function.
- $\varepsilon_{\text{crit}}$  is the critical activation energy for depassivation.
- $p$  is the exponent of the activation energy.
- $\gamma$  is the field enhancement prefactor.
- $\rho$  is the field enhancement exponent.
- $W_f$  is the activation energy for the depassivation process.
- $\alpha$  is the field-induced barrier-lowering factor.
- $\chi$  is the kinetic energy–induced barrier-lowering factor.

## Chapter 19: Degradation

### MSC–Hydrogen Transport Degradation Model

Similarly, the hydrogen depassivation rate induced by cold electrons can be written as:

$$c_{\text{PMI}} = \sigma v_{\text{th}} n \exp \left[ \gamma \frac{F}{V/\text{cm}^3} - \frac{\Theta(W_f - q\alpha F)(W_f - q\alpha F)}{kT} \right] \quad (609)$$

where  $v_{\text{th}}$  is the thermal velocity.

As the hydrogen passivation (electron emission) rate is not related to hot carriers, it is modeled simply as:

$$e = e_{\text{PMI}} \times \frac{[\text{H}]}{\text{1/cm}^3} \quad (610)$$

$$e_{\text{PMI}} = e_0 \exp \left[ -\frac{W_r}{kT} \right] \quad (611)$$

where  $e_0$  is the passivation rate prefactor, and  $W_r$  is the activation energy for the passivation process.

The capture-emission model `CEModel_Depassivation` implements [Equation 608](#) and [Equation 609](#) for electrons and holes. The additional index in the `CEModel_Depassivation` model selects the equation and the carrier type as follows:

```
MSConfig( ...
    Transition( ... CEModel("CEModel_Depassivation" 0))
                # Equation 609 for electrons
    Transition( ... CEModel("CEModel_Depassivation" 1))
                # Equation 609 for holes
    Transition( ... CEModel("CEModel_Depassivation" 2))
                # Equation 608 for electrons
    Transition( ... CEModel("CEModel_Depassivation" 3))
                # Equation 608 for holes
)
```

To use the `CEModel_Depassivation` model with the index 2 or 3 (hot electron–induced or hot hole–induced degradation), you must specify `eSHEDistribution` or `hSHEDistribution` in the `Physics` section to compute the electron or hole distribution function (see [Using Spherical Harmonics Expansion Method on page 857](#)).

The model coefficients and their defaults are given in [Table 111](#). The model coefficients can be changed in the `CEModel_Depassivation` section of the parameter file.

*Table 111 Default coefficients for `CEModel_Depassivation`*

Symbol	Parameter name (Electrons)	Default value (Electrons)	Parameter name (Holes)	Default value (Holes)	Unit
$\sigma$	xsec_e	$1.0 \times 10^{-24}$	xsec_h	$1.0 \times 10^{-24}$	$\text{cm}^2$
$v_{\text{th}}$	vth_e	$2.0 \times 10^7$	vth_h	$2.0 \times 10^7$	$\text{cm/s}$

*Table 111 Default coefficients for CEModel\_Depassivation (Continued)*

Symbol	Parameter name (Electrons)	Default value (Electrons)	Parameter name (Holes)	Default value (Holes)	Unit
$\gamma$	gamma_e	$2.7 \times 10^{-7}$	gamma_h	$2.7 \times 10^{-7}$	1
$\rho$	rho_e	1.0	rho_h	1.0	1
$\alpha$	alpha_e	$9.0 \times 10^{-9}$	alpha_h	$9.0 \times 10^{-9}$	cm
$\chi$	chi_e	1.0	chi_h	1.0	1
$W_f$	wf_e	0.5	wf_h	0.5	eV
$e_0$	e0_e	$3.0 \times 10^{-9}$	e0_h	$3.0 \times 10^{-9}$	1/s
$W_r$	Wr_e	0	Wr_h	0	eV
$d\varepsilon$	dE_e	0.01	dE_h	0.01	eV
$\varepsilon_{\text{crit}}$	Ecrit_e	0	Ecrit_h	0	eV
$p$	P_e	0	P_h	0	1
$\varepsilon_{\min}$	Emin_e	0	Emin_h	0	eV
$\varepsilon_{\max}$	Emax_e	5	Emax_h	5	eV

## Using MSC–Hydrogen Transport Degradation Model

You can select the regions and interfaces where the hydrogen transport equations are to be solved by specifying the keyword `HydrogenDiffusion` in the corresponding `Physics` section of the command file. For example:

```
Physics ( Material = "Oxide" ) {
    HydrogenDiffusion
}
```

The keyword `HydrogenDiffusion` can be specified with the `HydrogenReaction` arguments (see [Reactions Between Mobile Elements on page 614](#)).

## Chapter 19: Degradation

### MSC–Hydrogen Transport Degradation Model

Specifying HydrogenDiffusion in the interface-specific Physics section gives a surface recombination term determined by  $r_i$  and  $[X_i]_0$  at the corresponding interface.

To activate the transport of hydrogen atoms, hydrogen molecules, and hydrogen ions, the keywords HydrogenAtom, HydrogenMolecule, HydrogenIon, HydrogenSpeciesA, HydrogenSpeciesB, and HydrogenSpeciesC must be specified inside the Coupled statement of the Solve section.

For example, transient drift-diffusion simulation with transport of hydrogen atoms and hydrogen molecules can be specified by:

```
Solve {
    Transient(...) {
        Coupled { Poisson Electron Hole HydrogenAtom HydrogenMolecule }
    }
}
```

The parameters  $D_i$ ,  $E_{di}$ ,  $[X_i]_0$ , and  $r_i$  can be specified in the region-specific and interface-specific HydrogenDiffusion parameter set in the parameter file. For example:

```
Material = "Oxide" {
    HydrogenDiffusion {
        HydrogenAtom {
            d0 = 1.0e-13          # [cm^2*s^-1]
            Ed = 0                # [eV]
            atd = 1               # [1]
            n0 = 0                # [cm^-3]
            krec = 0              # [cm^-3*s^-1]
        }
        HydrogenMolecule {
            d0 = 1.0e-14          # [cm^2*s^-1]
            Ed = 0                # [eV]
            atd = 1               # [1]
            n0 = 0                # [cm^-3]
            krec = 0              # [cm^-3*s^-1]
        }
        HydrogenIon {
            ...
        }
    }
}

MaterialInterface = "Oxide/PolySi" {
    HydrogenDiffusion {
        HydrogenAtom {
            n0 = 1                # [cm^-3]
            krec = 1              # [cm^-2*s^-1]
        }
    }
}
```

## Chapter 19: Degradation

### Two-Stage NBTI Degradation Model

Here  $d_0$ ,  $E_d$ ,  $a_{td}$ ,  $n_0$ , and  $k_{rec}$  correspond to  $D_i$ ,  $E_{di}$ ,  $\alpha_{td}$ ,  $[X_i]_0$ , and  $r_i$ . The default values of  $D_i$ ,  $E_{di}$ ,  $[X_i]_0$ , and  $r_i$  are zero; while  $\alpha_{td} = 1$  by default.

The keywords for plotting the densities of hydrogen atoms, hydrogen molecules, and hydrogen ions are:

```
Plot { ...
    HydrogenAtom HydrogenMolecule HydrogenIon
    HydrogenSpeciesA HydrogenSpeciesB HydrogenSpeciesC
}
```

## Changing Charge and Hydrogen Composition of Species

You can change the charge and hydrogen composition of the individual species `HydrogenAtom`, `HydrogenMolecule`, and `HydrogenIon` in the parameter file as follows:

```
HydrogenDiffusion {
    HydrogenAtom {
        Charge = 3
        Hydrogen = 5
    }
}
```

### Note:

You cannot specify different compositions for different materials or regions.

The specified species compositions are taken into account in the hydrogen transport equation (see [Equation 599](#)), the corresponding reactions (see [Reactions Between Mobile Elements on page 614](#)), the reactions with MSC states (see [Reactions With Multistate Configurations on page 616](#)), and the Poisson equation.

---

## Two-Stage NBTI Degradation Model

Negative bias temperature instability (NBTI) refers to the generation of positive oxide charges and interface traps in MOS structures under negative gate bias at elevated temperature, which affects the threshold voltage and on-currents of PMOSFETs [\[12\]](#).

Sentaurus Device provides a two-stage NBTI degradation model [\[1\]\[2\]](#), which assumes that the NBTI degradation proceeds using a two-stage process:

- The first stage includes the creation of  $E'$  centers (dangling bonds in amorphous oxides) from their neutral oxygen vacancy precursors, the charging and discharging of  $E'$  centers, and the total annealing of  $E'$  centers to neutral oxygen vacancy precursors.
- The second stage considers the creation of poorly recoverable  $P_b$  centers (dangling bonds at silicon–oxide interfaces).

## Chapter 19: Degradation

### Two-Stage NBTI Degradation Model

In the two-stage NBTI degradation model, the involved energy levels and activation energies are distributed widely and are treated as random variables. A random sampling technique is used to obtain the average change of the interface charge.

#### Note:

As the two-stage NBTI model is based on the semiclassical carrier density at the semiconductor–insulator interface, it is not recommended to use the model with quantum-correction models.

---

## Formulation

The two-stage NBTI degradation model considers a special trap having four internal states:

- $s_1$  : Oxygen vacancy as a precursor state
- $s_2$  : Positive  $E'$  center
- $s_3$  : Neutral  $E'$  center
- $s_4$  : Fixed positive charge with a  $P_b$  center

Each NBTI trap is characterized by seven independent random variables:

- $E_1$  : Trap level of the precursor [eV] (by default,  $-1.14 \leq E_1 < -0.31$ )
- $E_2$  : Trap level of the  $E'$  center [eV] (by default,  $0.01 \leq E_2 < 0.3$ )
- $E_4$  : Trap level of the  $P_b$  center [eV] (by default,  $0.01 \leq E_4 < 0.5$ )
- $E_A$  : Barrier energy of a transition from  $s_3$  to  $s_1$  [eV] (by default,  $0.01 \leq E_A < 1.15$ )
- $E_B$  : Barrier energy of a transition from  $s_1$  to  $s_2$  [eV] (by default,  $0.01 \leq E_B < 1.15$ )
- $E_D$  : Barrier energy of a transition between  $s_2$  to  $s_4$  [eV] (by default,  
$$E_D = 1.46, (E_D^2 - E_D)^{1/2} = 0.44)$$
- $r$  : Uniform number between 0 and 1; when  $r < C$ , a transition from  $s_2$  to  $s_4$  is allowed  
(by default,  $C = 0.12$ )

$E_1$ ,  $E_2$ ,  $E_4$ ,  $E_A$ , and  $E_B$  follow the uniform distribution between their minimum and maximum values.  $E_D$  follows the Fermi-derivative distribution characterized by its average and standard deviation. The trap levels  $E_1$ ,  $E_2$ , and  $E_4$  are defined relative to the valence band energy.

## Chapter 19: Degradation

### Two-Stage NBTI Degradation Model

The state occupation probability  $s_i$  satisfies the normalization condition:

$$\sum_{i=1}^4 s_i = 1 \quad (612)$$

and the following rate equations:

$$\dot{s}_1 = -s_1 k_{12} + s_3 k_{31} \quad (613)$$

$$\dot{s}_2 = s_1 k_{12} - s_2 (k_{23} + k_{24}) + s_3 k_{32} + s_4 k_{42} \quad (614)$$

$$\dot{s}_3 = s_2 k_{23} - s_3 (k_{32} + k_{31}) \quad (615)$$

$$\dot{s}_4 = s_2 k_{24} - s_4 k_{42} \quad (616)$$

with the transition rates:

$$k_{12} = e_C^{n,1} + c_V^{p,1} \quad (617)$$

$$k_{23} = c_C^{n,2} + e_V^{p,2} \quad (618)$$

$$k_{32} = e_C^{n,2} + c_V^{p,2} \quad (619)$$

$$k_{31} = v_1 \exp(-E_A/kT) \quad (620)$$

$$k_{24} = v_2 \exp[-(E_D - \gamma F)/kT] \Theta(C - r) \quad (621)$$

$$k_{42} = v_2 \exp[-(E_D + \Delta E_D + \gamma F)/kT] \Theta(C - r) \quad (622)$$

where:

- $v_1$  and  $v_2$  are the attempt frequencies (by default,  $v_1 = 1.0 \times 10^{13} \text{ s}^{-1}$  and  $v_2 = 5.11 \times 10^{15} \text{ s}^{-1}$ ).
- $\gamma$  is the prefactor for the field-dependent barrier energy (by default,  $\gamma = 7.4 \times 10^{-8} \text{ cm} \cdot \text{eV/V}$ ).
- $\Delta E_D$  is the additional barrier energy for  $k_{42}$  (by default,  $\Delta E_D = 0 \text{ eV}$ ).

In addition, it is assumed that the hole occupation probability  $f_{it}^p$  of the  $P_b$  center in the state 4 is determined by:

$$f_{it}^p = (e_C^{n,4} + c_V^{p,4})(1 - f_{it}^p) - (c_C^{n,4} + e_V^{p,4})f_{it}^p \quad (623)$$

The electron emission rate and the hole capture rate for the state 1 are given by:

$$e_C^{n,1} = \sigma_n v_{th}^n N_C \exp[-H(E_C - E_1)/kT + \Theta(F_{c,n})(|F|/F_{c,n})^{\rho_n} - E_B/kT] \quad (624)$$

$$c_V^{p,1} = \sigma_p v_{th}^p p \exp[-H(E_V - E_1)/kT + \Theta(F_{c,p})(|F|/F_{c,p})^{\rho_p} - E_B/kT] \quad (625)$$

## Chapter 19: Degradation

### Two-Stage NBTI Degradation Model

where:

- $\sigma_n$  and  $\sigma_p$  are the electron and hole capture cross-sections (by default,  $\sigma_n = 1.08 \times 10^{-15} \text{ cm}^2$  and  $\sigma_p = 1.24 \times 10^{-14} \text{ cm}^2$ ).
- $v_{\text{th}}^n$  and  $v_{\text{th}}^p$  are the electron and hole thermal velocity (by default,  $v_{\text{th}}^n = 1.5 \times 10^7 \text{ cm/s}$  and  $v_{\text{th}}^p = 1.2 \times 10^7 \text{ cm/s}$ ).
- $F$  is the insulator electric field.
- $F_{c,n}$  and  $F_{c,p}$  are the critical electric field (by default,  $F_{c,n} = -1 \text{ V/cm}$  and  $F_{c,p} = 2.83 \times 10^6 \text{ V/cm}$ ).
- $\rho_n$  and  $\rho_p$  are the exponents of the field-dependent term (by default,  $\rho_n = 2$  and  $\rho_p = 2$ ).
- $H(x) = x\Theta(x)$ .

When the energy-dependent hole distribution is available (`hSHEDistribution` is activated in the semiconductor region), you can use the following expression for  $c_V^{p,1}$  instead of [Equation 625](#):

$$c_V^{p,1} = 2g_v\sigma_p \sum_0^{\infty} (\varepsilon) g(\varepsilon) f(\varepsilon) \exp[-H(E_V - E_1 - \varepsilon)/kT + \Theta(F_{c,p})(|F|/F_{c,p})^{\rho_p} - E_B/kT] d\varepsilon \quad (626)$$

The electron capture rate, the electron emission rate, the hole capture rate, and the hole emission rate for state 2 and state 4 are given by:

$$c_C^{n,i} = \sigma_n v_{\text{th}}^n n \exp[-H(E_i - E_C)/kT] \quad (627)$$

$$e_C^{n,i} = \sigma_n v_{\text{th}}^n N_C \exp[-H(E_C - E_i)/kT] \quad (628)$$

$$c_V^{p,i} = \sigma_p v_{\text{th}}^p p \exp[-H(E_V - E_i)/kT] \quad (629)$$

$$e_V^{p,i} = \sigma_p v_{\text{th}}^p N_V \exp[-H(E_i - E_V)/kT] \quad (630)$$

where  $i = 2$  or  $4$ .

## Using Two-Stage NBTI Model

You can activate the two-stage NBTI model by specifying the `NBTI` command in the interface-specific `Physics` section of the command file. For example:

```
Physics ( MaterialInterface = "Silicon/Oxide" ) {
    NBTI (
        Conc = 5.0e12                                # N_0 [ /cm^2 ]
        NumberOfSamples = 1000                         # N_sample [1]
        hSHEDistribution | -hSHEDistribution          # (off by default)
```

## Chapter 19: Degradation

Extended Nonradiative Multiphonon Model

```
    )  
}
```

where `Conc` represents the density of the precursor  $N_0$ , and `NumberOfSamples` represents the number of random samples  $N_{\text{sample}}$ . Sentaurus Device generates  $N_{\text{sample}}$  random configurations for each interface vertex.

Then, the interface charge density is obtained from the ensemble average as follows:

$$Q = Q_{\text{ox}} + Q_{\text{it}} = qN_0 s_2 + s_4 + qN_0 s_4 f_{\text{it}}^p \quad (631)$$

where  $x = \frac{1}{N_{\text{sample}}} \sum_{j=1}^{N_{\text{sample}}} x^j$ .

During transient simulation, state occupation probabilities will change following the kinetic equations, which will change the interface charge density. It is assumed that  $s_1 = 1$  at the beginning. In addition, you can restart NBTI degradation simulations by loading the saved data.

When the `hSHEDistribution` keyword is specified in the `NBTI` command (off by default),  $c_V^{p,1}$  is computed from [Equation 626](#) instead of [Equation 625](#).

**Note:**

You need to specify `hSHEDistribution` in the `Physics` section to compute the hole distribution function (see [Using Spherical Harmonics Expansion Method on page 857](#)).

You can plot the density of charge and the density of each state as follows:

```
Plot {  
    InterfaceNBTICharge    # [1/cm^2]  
    InterfaceNBTIState1    # [1/cm^2]  
    InterfaceNBTIState2    # [1/cm^2]  
    InterfaceNBTIState3    # [1/cm^2]  
    InterfaceNBTIState4    # [1/cm^2]  
}
```

The model parameters are defined in the interface-specific `NBTI` parameter set.

---

## Extended Nonradiative Multiphonon Model

The extended nonradiative multiphonon (eNMP) model [\[3\]](#) can be used to investigate degradation related to NBTI. The model accounts for the behavior of oxide defects (traps) that can have four internal states, two of which are metastable.

As with the two-stage NBTI degradation model, the eNMP model uses a random sampling technique to obtain the average behavior of multiple sample defects ( $N_{\text{sample}}$ ) at each

## Chapter 19: Degradation

### Extended Nonradiative Multiphonon Model

interface vertex. The sample defects are considered to have random insulator positions, and distributed energy levels and activation energies.

Each sample defect is assumed to be located at a distance  $x_t$  from the interface, where  $x_t$  is distributed randomly between a minimum and maximum location:

$$x_{t,\min} < x_t < x_{t,\max} \quad (632)$$

**Note:**

The implementation treats the average charge associated with the sample defects as an interface charge.

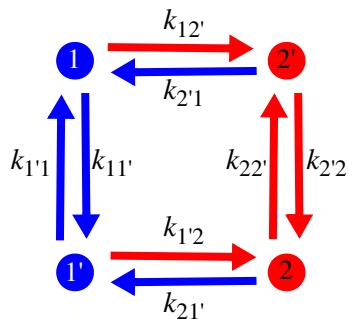
## eNMP Model Description

The eNMP model considers a special trap having four internal states:

- $s_1$ : Neutral stable state
- $s_2$ : Positive stable state
- $s_{1'}$ : Neutral metastable state
- $s_{2'}$ : Positive metastable state

The model describes the transitions between these states according to the diagram in [Figure 24](#), in which the  $k_{ij}$  terms represent the transition rates between the states.

*Figure 24 State diagram for the eNMP model*



The state occupation probabilities  $s_i$  satisfy the normalization condition:

$$s_1 + s_{1'} + s_2 + s_{2'} = 1 \quad (633)$$

The rate equations for the transitions are given by:

$$\dot{s}_1 = -s_1(k_{11'} + k_{12'}) + s_1k_{11'} + s_2k_{21'} \quad (634)$$

$$\dot{s}_2 = -s_2(k_{21'} + k_{22'}) + s_1k_{12'} + s_2k_{22'} \quad (635)$$

## Chapter 19: Degradation

### Extended Nonradiative Multiphonon Model

$$\dot{s}_{1'} = -s_{1'}(k_{1'1} + k_{1'2}) + s_1 k_{11'} + s_2 k_{21'} \quad (636)$$

$$\dot{s}_{2'} = -s_{2'}(k_{2'1} + k_{2'2}) + s_1 k_{12'} + s_2 k_{22'} \quad (637)$$

The eight transition rates are given by the following expressions:

$$k_{12'} = (1 + R_1)^{3/2} \sigma_p v_{\text{th}}^p p \exp(-\varepsilon_{12'}/kT) \quad (638)$$

$$k_{21'} = (1 + R_1)^{3/2} \sigma_p v_{\text{th}}^p N_V \exp(-\varepsilon_{12'}/kT) \exp(-(E_t - E_V - \varepsilon_{T2'})/kT) \quad (639)$$

$$k_{1'2} = (1 + R_{1'})^{3/2} \sigma_p v_{\text{th}}^p p \exp(-\varepsilon_{1'2}/kT) \quad (640)$$

$$k_{21'} = (1 + R_{1'})^{3/2} \sigma_p v_{\text{th}}^p N_V \exp(-\varepsilon_{1'2}/kT) \exp(-(E_t' - E_V)/kT) \quad (641)$$

$$k_{11'} = v_0 \exp(-(\varepsilon_{1'1} + E_t' - E_t)/kT) \quad (642)$$

$$k_{1'1} = v_0 \exp(-\varepsilon_{1'1}/kT) \quad (643)$$

$$k_{22'} = v_0 \exp(-(\varepsilon_{2'2} + \varepsilon_{T2'})/kT) \quad (644)$$

$$k_{2'2} = v_0 \exp(-\varepsilon_{2'2}/kT) \quad (645)$$

with:

$$\varepsilon_{12'} = \frac{(S_1 \hbar \omega_1)}{(1 + R_1)^2} + \frac{R_1}{1 + R_1} (E_V - E_t + \varepsilon_{T2'}) \quad (646)$$

$$\varepsilon_{1'2} = \frac{(S_1 \hbar \omega_{1'})}{(1 + R_{1'})^2} + \frac{R_{1'}}{1 + R_{1'}} (E_V - E_t') \quad (647)$$

For an insulator defect located a distance  $x_t$  from the interface:

$$\sigma_p = \sigma_{p0} \exp(-x_t/x_0) \quad (648)$$

$$E_t - E_V = E_{t0} - E_{V0} + qx_t|F| = \Delta E_t + qx_t|F| \quad (649)$$

$$E_t' - E_V = E_{t0'} - E_{V0} + qx_t|F| = \Delta E_t' + qx_t|F| \quad (650)$$

In these expressions:

- $\sigma_{p0}$  is the hole-capture cross section, and  $\exp(-x_t/x_0)$  accounts for the trap depth of tunneling.
- $\Delta E_t$  and  $\Delta E_t'$  are the energy levels of the defect in the neutral stable state and the neutral metastable state, respectively, relative to the valence band energy in the absence of an electric field.
- $F$  is the insulator electric field.
- $v_{\text{th}}^p$  is the hole thermal velocity.
- $\varepsilon_{ij}$  represent the transition energies between states.
- $\varepsilon_{T2'}$  is the energy of the positive metastable state relative to the positive stable state.

## Chapter 19: Degradation

### Extended Nonradiative Multiphonon Model

- $\omega_i$  are vibrational frequencies.
- $R_1 = \omega_1/\omega_2$  and  $R_{1'} = \omega_{1'}/\omega_2$ .
- $S_i \hbar \omega_i$  are relaxation energies, where  $S_i$  are known as Huang–Rhys factors.
- $v_0$  is the attempt frequency.

---

## Using the eNMP Model

You can activate the eNMP model in an interface-specific `Physics` section of the command file with the keyword `eNMP`. The command syntax and options are:

```
Physics (MaterialInterface="<mat1>/<mat2>" |
          RegionInterface="<reg1>/reg2>") {
    eNMP (
        NumberOfSamples = <Nsamp>      # Typically, 1000
        Conc = <N0>                      # 1/cm2, for example, 5e12
        [ SFactor=<dataset_name-or-pmi_model_name> " ]
        [ <eNMPTransitionRates_pmi_model_name> [StateCharge=<1 | -1>] ]
    )
}
```

where `Conc` represents the density of the precursor  $N_0$ . By default, the precursor concentration is constant over the interface. However, if `SFactor` is specified, the precursor concentration is obtained from a dataset (or a PMI user field that is read from the file specified with `PMIUserFields` in the `File` section of the command file), or from a space factor PMI written by users (see [Space Factor on page 1479](#)). If `SFactor` is specified without specifying `Conc`, the `SFactor` interface values are used directly. If `SFactor` is specified in addition to `Conc`, the `SFactor` values are normalized by the largest `SFactor` value and multiplied by `Conc`.

The `NumberOfSamples` keyword represents the number of random samples,  $N_{\text{sample}}$ . Sentaurus Device generates  $N_{\text{sample}}$  random defects for each interface vertex. Then, the interface charge density is obtained from the ensemble average of the charge states:

$$Q = qN_0 s_2 + s_2 \quad (651)$$

$$\text{where } x = \frac{1}{N_{\text{sample}}} \sum_{j=1}^{N_{\text{sample}}} x^j.$$

During transient simulations, the state occupation changes according to the rate equations ([Equation 634](#) to [Equation 637](#)), which will change the interface charge density. It is assumed that  $s_1 = 1$  at the beginning, with all other states unoccupied.

## eNMP Quantities Available for Plotting

Plots of the interface charge density and the density of each state can be specified in the Plot section of the command file:

```
Plot {
    InterfaceeNMPCharge
    InterfaceeNMPState1
    InterfaceeNMPState2
    InterfaceeNMPState1p
    InterfaceeNMPState2p
}
```

You can also obtain the plots of the capture and emission times:

```
Plot {
    eNMPCaptureTime
    eNMPEmissionTime
}
```

For this purpose, Sentaurus Device uses the first passage times given in [3]:

$$\frac{1}{\tau_{cap}} = \frac{1}{\tau_{cap}^{1'}} + \frac{1}{\tau_{cap}^{2'}} \quad (652)$$

$$\frac{1}{\tau_{em}} = \frac{1}{\tau_{em}^{1'}} + \frac{1}{\tau_{em}^{2'}} \quad (653)$$

where:

$$\frac{1}{\tau_{cap}^{1'}} = \frac{k_{11}k_{1'2}}{k_{11'} + k_{1'2} + k_{1'1}} \quad (654)$$

$$\frac{1}{\tau_{cap}^{2'}} = \frac{k_{12}k_{2'2}}{k_{12'} + k_{2'2} + k_{2'1}} \quad (655)$$

$$\frac{1}{\tau_{em}^{1'}} = \frac{k_{21}k_{1'1}}{k_{21'} + k_{1'1} + k_{1'2}} \quad (656)$$

$$\frac{1}{\tau_{em}^{2'}} = \frac{k_{22}k_{2'1}}{k_{22'} + k_{2'1} + k_{2'2}} \quad (657)$$

## Chapter 19: Degradation

Extended Nonradiative Multiphonon Model

### eNMP Model Parameters

The parameters used in the eNMP model are defined in the interface-specific eNMP parameter set. [Table 112](#) to [Table 115](#) list the default parameter values for silicon–oxide interfaces.

*Table 112 eNMP parameters that follow a Gaussian distribution: Default coefficients for silicon–oxide interfaces*

Symbol	Parameter name	Mean	Standard	Unit
$\Delta E_t$	E <sub>t</sub>	-0.5	0.1	eV
$\Delta E_t'$	E <sub>tp</sub>	0.5	0.1	eV
$R_1$	R	0.6	0.1	1
$R_{1'}$	R <sub>p</sub>	0.6	0.1	1
$S_1 \hbar \omega_1$	E <sub>S</sub>	1.0	0.1	eV
$S_{1'} \hbar \omega_{1'}$	E <sub>Sp</sub>	1.0	0.1	eV
$\varepsilon_{T2'}$	E <sub>T2p</sub>	0.5	0.1	eV
$\varepsilon_{1'1}$	E <sub>1p1</sub>	1.0	0.1	eV
$\varepsilon_{2'2}$	E <sub>2p2</sub>	0.5	0.1	eV

*Table 113 eNMP parameters that follow a uniform distribution: Default coefficients for silicon–oxide interfaces*

Symbol	Parameter name	Minimum	Maximum	Unit
$x_t$	x <sub>t</sub>	0.0	8.0	Å

*Table 114 eNMP parameters having separate values for electrons and holes: Default coefficients for silicon–oxide interfaces*

Symbol	Parameter name	Electrons	Holes	Unit
$\sigma_0$	xsec	$1.0 \times 10^{-15}$	$1.0 \times 10^{-15}$	cm <sup>2</sup>
$v_{th}$	v <sub>th</sub>	$1.5 \times 10^7$	$1.2 \times 10^7$	cm/s
$x_0$	x <sub>0</sub>	0.5	0.5	Å

*Table 115 eNMP parameters having a single value: Default coefficients for silicon–oxide interfaces*

Symbol	Parameter name	Value	Unit
$v_0$	$\nu_{00}$	$1.0 \times 10^{13}$	s <sup>-1</sup>

## eNMP Transition Rates PMI Model

An eNMP transition rates PMI model (see [eNMP Transition Rates on page 1402](#)) created by users can be utilized to calculate the transition rates for the eNMP model. This PMI allows for the possibility of either a positive or negative state charge, and provides additional dependencies that are not included in the built-in eNMP model.

---

## Hot-Carrier Stress Degradation Model

The hot-carrier stress (HCS) degradation model is based on work in [\[15\]](#), and can be used as a general degradation model for MOS-based devices.

The HCS degradation model includes different mechanisms that contribute to bond breakage and the formation of interface traps [\[11\]\[16\]\[17\]](#):

- Single-particle (SP) processes, where a single particle is responsible for bond breakage
- Multiple-particle (MP) processes, where the combined actions of several particles contribute to bond breakage
- Field-enhanced thermal (TH) processes, where thermal interactions with the lattice contribute to bond breakage

Sentaurus Device includes both an electron version of the model (`eHCSDegradation`) and a hole version of the model (`hHCSDegradation`).

The model and its usage are described in the following sections.

---

## Model Description

The model equations are presented here. Details can be found in [\[15\]](#). The equations presented apply to both the `eHCSDegradation` and `hHCSDegradation` models.

## Single-Particle and Multiple-Particle Interface-Trap Densities

For the SP case, the interface trap density as a function of time and activation energy is given by

$$N_{it,SP}(\mathbf{r}, t, E_{SP}) = P_{SP} N_0 [1 - e^{-k_{SP}(\mathbf{r}, E_{SP})t}] \quad (658)$$

where:

- $P_{SP}$  is the probability for defect generation by SP processes.
- $N_0$  is the maximum number of interface bonds.
- $E_{SP}$  is the activation energy for SP processes.
- $k_{SP}(\mathbf{r}, E_{SP})$  is the reaction rate for SP processes.

For the MP case, the interface trap density is given by:

$$N_{it,MP}(\mathbf{r}, t, E_{MP}) = P_{MP} N_0 \left[ \frac{P_{emi}}{P_{pass}} \frac{P_u}{P_d} \frac{N_l}{N_i} (1 - e^{-P_{emi}t}) \right]^{1/2} \quad (659)$$

where:

- $P_{MP}$  is the probability for defect generation by MP processes.
- $N_l$  is the number of energy levels in the oscillator that models the bond.

The emission and passivation probabilities  $P_{emi}$  and  $P_{pass}$  are modeled as Arrhenius laws:

$$P_{emi} = v_{emi} e^{-E_{emi}/(k_B T)} \quad (660)$$

$$P_{pass} = v_{pass} e^{-E_{pass}/(k_B T)} \quad (661)$$

where:

- $v_{emi}$  and  $v_{pass}$  are the emission and passivation frequencies, respectively.
- $E_{emi}$  and  $E_{pass}$  are the emission and passivation energies, respectively.

The oscillator excitation and de-excitation probability rates are given by:

$$P_u = k_{ph} e^{-E_{ph}/(k_B T)} + k_{MP}(\mathbf{r}, E_{MP}) \quad (662)$$

$$P_d = k_{ph} + k_{MP}(\mathbf{r}, E_{MP}) \quad (663)$$

where:

- $E_{ph}$  and  $k_{ph}$  are the phonon energy and the reaction rate, respectively.
- $E_{MP}$  is the activation energy for MP processes.
- $k_{MP}(\mathbf{r}, E_{MP})$  is the reaction rate for MP processes.

## Chapter 19: Degradation

### Hot-Carrier Stress Degradation Model

The reaction rates for SP and MP processes are given by scattering-rate integrals:

$$k_{\text{SP}}(\mathbf{r}, E_{\text{SP}}) = \int_{E_{\text{SP}}}^{\infty} f(\mathbf{r}, E) g(E) v(E) \sigma_{\text{SP}}(E) dE \quad (664)$$

$$k_{\text{MP}}(\mathbf{r}, E_{\text{MP}}) = \int_{E_{\text{MP}}}^{\infty} f(\mathbf{r}, E) g(E) v(E) \sigma_{\text{MP}}(E) dE \quad (665)$$

where:

- $f(\mathbf{r}, E)$  is the carrier distribution function.
- $g(E)$  is the total density-of-states.
- $v(E)$  is the magnitude of the carrier velocity.
- $\sigma_{\text{SP}}(E)$  and  $\sigma_{\text{MP}}(E)$  are the SP and MP reaction cross-sections.

The reaction cross-sections are given by:

$$\sigma_{\text{SP}}(E) = \sigma_{\text{SP}0} \left[ \frac{E - E_{\text{SP}}}{k_B T} \right]^{p_{\text{SP}}} \quad (666)$$

$$\sigma_{\text{MP}}(E) = \sigma_{\text{MP}0} \left[ \frac{E - E_{\text{MP}}}{k_B T} \right]^{p_{\text{MP}}} \quad (667)$$

where:

- $p_{\text{SP}}$  and  $p_{\text{MP}}$  are exponents characterizing the SP and MP processes.
- $\sigma_{\text{SP}0}$  and  $\sigma_{\text{MP}0}$  are fitting parameters.

## Field-Enhanced Thermal Degradation

The interface-trap density due to field-enhanced thermal degradation is given by:

$$N_{\text{it,TH}}(\mathbf{r}, t, E_{\text{TH}}) = P_{\text{TH}} N_0 [1 - e^{-k_{\text{TH}}(E_{\text{TH}})t}] \quad (668)$$

where:

- $P_{\text{TH}}$  is the probability for defect generation by thermal processes.
- The reaction rate for bond breakage  $k_{\text{TH}}$  is given by:

$$k_{\text{TH}}(\mathbf{r}, E_{\text{TH}}) = v_{\text{TH}} e^{-E_{\text{TH}}/(k_B T)} \quad (669)$$

$$E_{\text{TH}} = E_{\text{TH}0} - p E_{\text{ox}} \quad (670)$$

where:

- $v_{\text{TH}}$  is the lattice collision frequency.

- $E_{TH0}$  is the activation energy for thermal processes in the absence of oxide field  $E_{ox}$ .
- $p$  is the effective dipole moment.

## Carrier Distribution Function

There are two options for obtaining the carrier distribution function  $f(\mathbf{r}, E)$  used in the calculation of the scattering-rate integrals ([Equation 664](#) and [Equation 665](#)):

- From the spherical harmonics expansion (SHE) method
- From an approximate analytic formulation

### Spherical Harmonics Expansion Option

The SHE method computes the microscopic carrier-energy distribution function by solving the lowest-order SHE of the Boltzmann transport equation. When the carrier distribution function is available through the SHE method, it can optionally be used directly in the evaluation of [Equation 664](#) and [Equation 665](#). In this case, the band structure quantities  $g(E)$  and  $v(E)$  used in these equations are the same as those used in the SHE calculations.

#### Note:

If the carrier distribution function is available through the SHE method, the SHE temperature will be used instead of the carrier temperature in the model equations, if the hydrodynamic model is not used.

### Approximate Analytic Option

A method for obtaining an approximate carrier distribution function [\[15\]](#) is available. In this case,  $f(\mathbf{r}, E)$  is modeled using an analytic non-Maxwellian formulation:

$$f_0(\mathbf{r}, E) = \frac{1}{A(\mathbf{r})} \exp\left[-\alpha(\mathbf{r}) \frac{\gamma(E)}{k_B T_c(\mathbf{r})}\right] \quad (671)$$

$$\gamma(E) = \frac{E(1 + \delta E)}{1 + \beta E} \quad (672)$$

where:

- $T_c(\mathbf{r})$  is the carrier temperature ( $T_n$  for eHCSDegradation and  $T_p$  for hHCSDegradation).
- $\delta$  and  $\beta$  are fitting parameters.

## Chapter 19: Degradation

### Hot-Carrier Stress Degradation Model

- $A(\mathbf{r})$  and  $\alpha(\mathbf{r})$  are position-dependent parametric factors that are determined by requiring that:

$$c(\mathbf{r}) = \int_0^{E_{\max}} f_0(\mathbf{r}, E) g(E) dE \quad (673)$$

$$T_c(\mathbf{r}) = \frac{2}{3k_B} \frac{\int_0^{E_{\max}} E f_0(\mathbf{r}, E) g(E) dE}{c(\mathbf{r})} \quad (674)$$

where  $c(\mathbf{r})$  is the electron concentration for eHCSDegradation and the hole concentration for hHCSDegradation.  $E_{\max} = 10$  eV is used as the upper integration limit.

## Bond Dispersion

The reaction rates given by [Equation 664](#), [Equation 665](#), and [Equation 669](#) are for a discrete activation energy. As an option, these rates can be modified to account for bond dispersion by assuming a distribution function for the activation energies:

$$g_{A,j}(E) = \frac{1}{\sigma_g} \frac{\exp\left[\frac{E_j - E}{\sigma_g}\right]}{1 + \exp\left[\frac{E_j - E}{\sigma_g}\right]^2} \quad (675)$$

where  $\sigma_g$  is a dispersion width and  $j = \text{SP, MP, or TH}$ .

The interface trap generation for each process  $j$  is then calculated from:

$$N_{it,j}(\mathbf{r}, t) = \frac{1}{N_{g_{A,j}}} \int_{E_j - m\sigma_g}^{E_j + m\sigma_g} g_{A,j}(E) N_{it,j}(\mathbf{r}, t, E) dE \quad (676)$$

where:

$$N_{g_{A,j}} = \int_{E_j - m\sigma_g}^{E_j + m\sigma_g} g_{A,j}(E) dE \quad (677)$$

Note that [Equation 676](#) requires the scattering rates for each energy used in the calculation of the integral:

$$k_{SP/MP}(\mathbf{r}, E) = \int_E^{\infty} f(\mathbf{r}, E') g(E') v(E') \sigma_{SP}(E') dE' , E_{SP/MP} - m\sigma_g < E < E_{SP/MP} + m\sigma_g \quad (678)$$

$$k_{TH}(\mathbf{r}, E) = v_{TH} e^{-E/(k_B T)} , E_{TH} - m\sigma_g < E < E_{TH} + m\sigma_g \quad (679)$$

## Chapter 19: Degradation

### Hot-Carrier Stress Degradation Model

The total interface-trap generation rate is taken as the sum of the separate processes:

$$N_{it}(\mathbf{r}, t) = N_{it,SP}(\mathbf{r}, t) + N_{it,MP}(\mathbf{r}, t) + N_{it,TH}(\mathbf{r}, t) \quad (680)$$

---

## Using the HCS Degradation Model

You can include the HCS degradation model in a transient simulation by specifying the `eHCSDegradation` option, or the `hHCSDegradation` option, or both options as part of a `Traps` specification for the interface of interest.

The syntax and options for these models are:

```
Physics (MaterialInterface = "Silicon/Oxide") {
    Traps (
        (eHCSDegradation([SHE] [BondDispersion] [BDEnergyIntervals=<int>])
         # Specify appropriate parameters for the generated traps.
         Acceptor Level EnergyMid=0 fromMidBandGap
         )
        (hHCSDegradation([SHE] [BondDispersion] [BDEnergyIntervals=<int>])
         # Specify appropriate parameters for the generated traps.
         Donor Level EnergyMid=0 fromMidBandGap
         )
    )
}
```

Notes regarding this specification:

- If the `SHE` option is selected, the distribution function and the band-structure quantities are obtained from the `SHE` method specified in the `Physics` section. For example:

```
Physics (Material = "Silicon") {
    eSHEDistribution(FullBand)
}
```

- If `SHE` is not selected, an approximate distribution function is used as described in [Approximate Analytic Option on page 634](#). In this case, the band-structure quantities used in the calculations will be taken from the `SHE` full-band structure files.
- `BondDispersion` is used by default. Use `-BondDispersion` to switch off this option. The number of energy intervals to use in the bond dispersion integral ([Equation 676](#)) can be specified with `BDEnergyIntervals` (default: 20).
- If using `FixedCharge` traps, negative charge will be created with the `eHCSDegradation` model, and positive charge will be created with the `hHCSDegradation` model.

Parameters used in the model can be accessed through the `HCSDegradation` parameter set in the parameter file. Since this model describes interface trap generation, these parameters will only be recognized if they are part of a `MaterialInterface` or

## Chapter 19: Degradation

### Hot-Carrier Stress Degradation Model

RegionInterface specification. [Table 116](#) lists the default parameter values for silicon–oxide interfaces.

*Table 116 HCS degradation model: Default coefficients for silicon–oxide interfaces*

Symbol	Parameter name	Electrons	Holes	Unit
$P_{SP}$	Prsp	1.0	1.0	1
$P_{MP}$	Prmp	1.0	1.0	1
$P_{TH}$	Prth	1.0	1.0	1
$N_0$	N0	$1.0 \times 10^{12}$	$1.0 \times 10^{12}$	$\text{cm}^{-2}$
$k_{ph}$	kph	$1.0 \times 10^8$	$1.0 \times 10^8$	$\text{s}^{-1}$
$E_{ph}$	Eph	0.25	0.25	eV
$E_{emi}$	Eemi	0.26	0.26	eV
$E_{pass}$	Epass	0.2	0.2	eV
$E_{TH0}$	Eth0	1.9	1.9	eV
$N_l$	Nlev	10	10	1
$v_{emi}$	nu_emi	$1.0 \times 10^{12}$	$1.0 \times 10^{12}$	$\text{s}^{-1}$
$v_{pass}$	nu_pass	$1.0 \times 10^{12}$	$1.0 \times 10^{12}$	$\text{s}^{-1}$
$v_{th}$	nu_th	$1.0 \times 10^{13}$	$1.0 \times 10^{13}$	$\text{s}^{-1}$
$p$	p	15.5	15.5	eÅ
$E_{SP}$	Esp	3.1	3.1	eV
$E_{MP}$	Emp	0.25	0.25	eV
$P_{SP}$	psp	11	11	1
$P_{MP}$	pmp	0.1	0.1	1
$\sigma_{SP0}$	Xsecsp	$3.0 \times 10^{-29}$	$3.0 \times 10^{-29}$	$\text{cm}^2$
$\sigma_{MP0}$	Xsecmp	$1.0 \times 10^{-24}$	$1.0 \times 10^{-24}$	$\text{cm}^2$
$\sigma_g$	sigmag	0.100	0.100	eV
$m$	m	3	3	1
$\delta$	delta	1.0	1.0	$\text{eV}^{-1}$
$\beta$	beta	0.15	0.15	$\text{eV}^{-1}$

## Activated Barrier Double Well Thermionic Model

One contribution to the overall threshold voltage shift during and after negative bias temperature instability (NBTI) or positive bias temperature instability (PBTI) is due to hole or electron trapping and detrapping from process-dependent preexisting traps in the gate insulator. This effect can be described with the activated barrier double well thermionic (ABDWT) model [18].

### Model Description

The ABDWT model considers two energy levels  $E_1$  and  $E_2$  that are separated by a barrier  $E_B$ . The barrier  $E_B$  is thermally activated and lowers by  $\Delta = \gamma E_{\text{ins}}$  with applied bias, while the energy level  $E_2$  lowers by  $m\Delta = m\gamma E_{\text{ins}}$  with the fit factors  $\gamma$  and  $m$ , and the insulator electric field  $E_{\text{ins}}$ .

The ABDWT model considers a special trap having two internal states:

- $s_1$ : Uncharged state at  $E_1$
- $s_2$ : Charged state at  $E_2$

The ABDWT model treats the two energy levels as being connected thermally overcoming an activation energy. In real devices, traps are also located in the bulk of the gate insulator where carriers can be captured and emitted additionally through tunneling processes. The ABDWT model is an effective model, meaning that both connecting processes are considered in an effective manner by fit factors and distributions for  $E_B$  and  $E_2$ .

The barrier energy  $E_B$  is distributed according to a Gaussian distribution where a temperature dependence is introduced for the mean energy barrier  $E_B(T)$  and its variance  $\sigma_{E_B}(T)$  according to:

$$E_B(T) = E_{B0} \cdot e^{-\frac{E_A}{kT}} \quad (681)$$

$$\sigma_{E_B}(T) = \sigma_{E_{B0}} \cdot e^{-\frac{E_A}{kT}}$$

with the mean energy barrier  $E_{B0}$ , the variance  $\sigma_{E_{B0}}$ , and the activation energy  $E_A$ .

The energy level  $E_2$  is distributed according to a Gaussian distribution with the mean energy level  $E_2$  and its variance  $\sigma_{E_2}$ .

The rate equations are given by:

$$\dot{s}_1 = -s_1 k_{12} + s_2 k_{21} \quad (682)$$

$$\dot{s}_2 = s_1 k_{12} - s_2 k_{21} \quad (683)$$

## Chapter 19: Degradation

### Activated Barrier Double Well Thermionic Model

where the state occupation probability satisfies the normalization condition  $s_1 + s_2 = 1$ . The rate constants for the forward and backward transitions are given by:

$$k_{12} = v \cdot e^{-\frac{E_B - E_1 - \gamma E_{ins}}{kT}} \quad (684)$$

$$k_{21} = v \cdot e^{-\frac{E_B - (E_2 - m\gamma E_{ins}) - \gamma E_{ins}}{kT}} \quad (685)$$

with the attempt rate  $v$  and the reference energy level of the uncharged state  $E_1$ .

---

## Using the ABDWT Model

You activate the ABDWT model by specifying the `ABDWT` command in the interface-specific `Physics` section of the command file. For example:

```
Physics ( MaterialInterface = "Silicon/Oxide" ) {
    ABDWT (
        Conc = 5.0e18           # N_0 in insulator [1/cm^3]
        NumberOfSamples = 100
    )
}
```

where `Conc` sets the insulator trap concentration and `NumberOfSamples` sets the number of random samples. Sentaurus Device generates random configurations for each interface vertex.

Then, the interface states are obtained per interface vertex  $v$  from the ensemble average according to:

$$s_2 = \frac{1}{N_{sample}} \sum_{j=0}^{N_{sample}} s_{2,j}^v \quad (686)$$

During a transient simulation, state occupations change following the kinetic equations, which change the interface ABDWT states  $s_1$  and  $s_2$ . It is assumed that the uncharged state  $s_1$  is fully occupied at the beginning of the time evolution.

You can plot the resulting position-dependent interface-specific quantities as follows:

```
Plot {
    InterfaceABDWTState1   # uncharged state 1
    InterfaceABDWTState2   # charged state 2
}
```

For a more realistic handling of electrostatics, ABDWT charges are distributed into the insulator bulk.

## Chapter 19: Degradation

### Activated Barrier Double Well Thermionic Model

For this purpose, Sentaurus Device searches for all ABDWT-activated interfaces that are connected to the same insulator and computes the interface-averaged ABDWT state density  $s_2$ , which is used for the allocation of the ABDWT charge density inside the insulator volume according to:

$$q_{ABDWT} = N_0 s_2 \quad (687)$$

You can plot the resulting ABDWT charge density as follows:

```
Plot {
    ABDWTCharge # cm^-3
}
```

#### Note:

ABDWT charges are considered only in the Poisson equation.

The model parameters are defined in the interface-specific ABDWT parameter set (see [Table 117](#)). For example:

```
ABDWT {
    nu      = 1.4e14
    gamma   = 5.0e-9
    m       = 2.0
    EBMean  = 1.3
    EA      = 0.02
    EBSpread = 0.2
    E2Mean  = 0.2
    E2Spread = 0.07
}
```

*Table 117 ABDWT model parameters*

Symbol	Parameter name	Default value	Unit
v	nu	1.0e13	1/s
$\gamma$	gamma	4.0e-9	e · cm
m	m	2.0	1
$E_{B0}$	EBMean	1.3	eV
$E_A$	EA	0.02	eV
$\sigma_{E_{B0}}$	EBSpread	0.2	eV
$E_2$	E2Mean	0.2	eV
$\sigma_{E_2}$	E2Spread	0.07	eV

---

## Transient Trap Occupancy Model

Trap occupancy of interface traps is computed using the transient trap occupancy model. It is expected that only those traps that are above the Fermi level, as in the case of negative bias temperature instability (NBTI), contribute to the threshold voltage ( $V_t$ ) degradation [19]. The interface traps below the Fermi level capture electrons in the case of NBTI and, therefore, are neutralized. While the traps physically exist, they do not contribute to  $V_t$  degradation.

---

### Model Description

The transient trap occupancy model uses the ABDWT model formulation to compute the charge occupancy. The double-well energy model, as described in [Activated Barrier Double Well Thermionic Model on page 638](#), considers a barrier between two energy wells, which carriers must cross thermionically to occupy the trapping site, that is, the charge state [19]. The solution of the rate equations for forward and backward fluxes gives the net occupancy of the charge state. The barrier and charge-state energies are assumed to follow the distribution of energies. Sentaurus Device generates random configurations of barriers and charge-state energies for each interface vertex.

Following the ABDWT model, the trap occupancy can be computed as an ensemble average over the distribution of energies as:

$$\text{Trap Occupancy} = \frac{1}{N_{\text{samples}}} \sum_{j=0}^{N_{\text{samples}}} s_{2,j}^v \quad (688)$$

Here,  $N_{\text{samples}}$  is the number of Gaussian samples, and  $s_{2,j}^v$  is the occupancy of the trap state at a given interface vertex,  $v$ , and for a given barrier and charge-state energy.

---

### Using the Transient Trap Occupancy Model

You can activate the model by specifying `TTOM` as an attribute to the state specification in `MSC` for an interface-specific `Physics` section of the command file as:

```
Physics (MaterialInterface="Oxide/Silicon") {
    MSConfigs (
        MSConfig (
            Name="msc0"
            State(Name="s0" Charge=0 Hydrogen=1 TTOM)
            State(Name="s1" Charge=1 Hydrogen=0 TTOM)
            Transition (
                Name="Depassivation" CEModel("CEModel_PMI",1)
                To="s1" From="s0"
                Reservoirs("VB"(Particles=+1) "HydrogenAtom"(Particles=-1))
```

## Chapter 19: Degradation

### Transient Trap Occupancy Model

```
)  
)  
}  
}
```

Trap occupancy applies to the state for which the TTOM keyword is specified (see [Table 317 on page 1691](#).

You can plot the fraction of traps above the Fermi level by using the following keyword in the Plot section:

```
Plot {  
    TTOMFraction      # Fraction of occupied traps  
}
```

The model parameters are defined in the interface-specific TTOM parameter set in the parameter file as:

```
TTOM {  
    nu = 1.4e14  
    gamma = 5.0e-9  
    m = 2.0  
    EBMean = 1.3  
    EA = 0.02  
    EBSpread = 0.2  
    E2Mean = 0.2  
    E2Spread = 0.07  
    NumberOfSamples = 100  
}
```

where:

- nu, gamma, and m are the model parameters.
- EBMean and EBSpread are the average and the standard deviation of the barrier energy, respectively.
- E2Mean and E2Spread are the average and the standard derivation of the charge-state energy, respectively.
- EA is the energy of the uncharged state of the double-well model.
- NumberOfSamples is the number of random Gaussian samples for the energy distribution.

*Table 118 Parameters of transient trap occupancy model*

Parameter name	Default value	Unit
Nu	$1.0 \times 10^{13}$	1/s
Gamma	$4.5 \times 10^{-9}$	e · cm

## Chapter 19: Degradation

### References

Table 118 Parameters of transient trap occupancy model (Continued)

Parameter name	Default value	Unit
m	3.2	1
EA	0	eV
EBSmean	1.1	eV
EBSspread	0.24	eV
E2Mean	0.18	eV
E2Spread	0.07	eV
NumberOfSamples	1	1

## References

- [1] T. Grasser *et al.*, “A Two-Stage Model for Negative Bias Temperature Instability,” in *IEEE International Reliability Physics Symposium (IRPS)*, Montréal, Québec, Canada, pp. 33–44, April 2009.
- [2] W. Goes *et al.*, “A Model for Switching Traps in Amorphous Oxides,” in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, San Diego, CA, USA, pp. 159–162, September 2009.
- [3] W. Gös, *Hole Trapping and the Negative Bias Temperature Instability*, PhD thesis, Technischen Universität Wien, Vienna, Austria, December 2011.
- [4] A. Plonka, *Time-Dependent Reactivity of Species in Condensed Media*, Lecture Notes in Chemistry, vol. 40, Berlin: Springer, 1986.
- [5] C. Hu *et al.*, “Hot-Electron-Induced MOSFET Degradation—Model, Monitor, and Improvement,” *IEEE Journal of Solid-State Circuits*, vol. SC-20, no. 1, pp. 295–305, 1985.
- [6] O. Penzin *et al.*, “MOSFET Degradation Kinetics and Its Simulation,” *IEEE Transactions on Electron Devices*, vol. 50, no. 6, pp. 1445–1450, 2003.
- [7] K. Hess *et al.*, “Theory of channel hot-carrier degradation in MOSFETs,” *Physica B*, vol. 272, no. 1–4, pp. 527–531, 1999.
- [8] Z. Chen *et al.*, “On the Mechanism for Interface Trap Generation in MOS Transistors Due to Channel Hot Carrier Stressing,” *IEEE Electron Device Letters*, vol. 21, no. 1, pp. 24–26, 2000.

## Chapter 19: Degradation

### References

- [9] B. Tuttle and C. G. Van de Walle, "Structure, energetics, and vibrational properties of Si-H bond dissociation in silicon," *Physical Review B*, vol. 59, no. 20, pp. 12884–12889, 1999.
- [10] M. A. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," *Microelectronics Reliability*, vol. 45, no. 1, pp. 71–81, 2005.
- [11] J. W. McPherson, R. B. Khamankar, and A. Shanware, "Complementary model for intrinsic time-dependent dielectric breakdown in  $\text{SiO}_2$  dielectrics," *Journal of Applied Physics*, vol. 88, no. 9, pp. 5351–5359, 2000.
- [12] J. H. Stathis and S. Zafar, "The negative bias temperature instability in MOS devices: A review," *Microelectronics Reliability*, vol. 46, no. 2-4, pp. 270–286, 2006.
- [13] A. E. Islam *et al.*, "Recent Issues in Negative-Bias Temperature Instability: Initial Degradation, Field Dependence of Interface Trap Generation, Hole Trapping Effects, and Relaxation," *IEEE Transactions on Electron Devices*, vol. 54, no. 9, pp. 2143–2154, 2007.
- [14] T. Grasser, W. Göts, and B. Kaczer, "Dispersive Transport and Negative Bias Temperature Instability: Boundary Conditions, Initial Conditions, and Transport Models," *IEEE Transactions on Device and Materials Reliability*, vol. 8, no. 1, pp. 79–97, 2008.
- [15] S. Reggiani *et al.*, "TCAD Simulation of Hot-Carrier and Thermal Degradation in STI-LDMOS Transistors," *IEEE Transactions on Electron Devices*, vol. 60, no. 2, pp. 691–698, 2013.
- [16] A. Bravaix *et al.*, "Hot-Carrier Acceleration Factors for Low Power Management in DC-AC stressed 40nm NMOS node at High Temperature," in *Proceedings of the 47th Annual International Reliability Physics Symposium (IRPS)*, Montréal, Québec, Canada, pp. 531–548, April 2009.
- [17] I. Starkov *et al.*, "Hot-carrier degradation caused interface state profile—Simulation versus experiment," *Journal of Vacuum Science & Technology B*, vol. 29, no. 1, p. 01AB09, 2011.
- [18] N. Choudhury *et al.*, "A Model for Hole Trapping-Detrapping Kinetics During NBTI in p-Channel FETs," in *4th IEEE Electron Devices Technology and Manufacturing Conference (EDTM)*, Penang, Malaysia, pp. 1–4, April 2020.
- [19] S. Mahapatra (ed.), *Fundamentals of Bias Temperature Instability in MOS Transistors: Characterization Methods, Process and Materials Impact, DC and AC Modeling*, Springer Series in Advanced Microelectronics, New Delhi: Springer, 2016.

# 20

## Organic Devices

---

This chapter describes the organic models available in Sentaurus Device.

The electrical conduction process in organic materials is different from crystal lattice semiconductors. However, similar concepts in semiconductor transport theory can be used in treating the conduction process in organic materials.

---

### Introduction to Organic Device Simulation

An organic material or semiconductor is formed from molecule chains, and the primary transport of carriers (electrons and holes) is through a *hopping* process. The lowest unoccupied molecular orbital (LUMO) and highest occupied molecular orbital (HOMO) energy levels in organic materials are analogous to the conduction and valence bands, respectively. In addition, excitons need to be considered since these bound electron–hole pairs contribute to the distribution of electron and hole populations. Traps are also central to organic transport, and these need to be taken into account appropriately.

Therefore, a combination of semiconductor models and organic physics models is required to model reasonably the physical transport processes of organic devices within the framework of Sentaurus Device. The following models are needed in a typical organic device simulation:

- The Poole–Frenkel mobility model is used to model the hopping transport of the carriers (electrons and holes). This model is dependent on electric field and temperature. It is common for electrons to have two orders of magnitude higher mobility than holes (see [Poole–Frenkel Mobility on page 455](#)).
- Organic–organic heterointerface physics requires special treatment for the ballistic transport of carriers and bulk excitons across heterointerfaces with energy barriers (see [Gaussian Transport Across Organic Heterointerfaces on page 873](#)).
- Gaussian density-of-states (DOS) approximates the effective DOS for electrons and holes in disordered organic materials and semiconductors (see [Gaussian Density-of-States for Organic Semiconductors on page 322](#)).

## Chapter 20: Organic Devices

### Introduction to Organic Device Simulation

- The traps model must be initialized with the appropriate capture cross sections and densities (see [Chapter 17 on page 543](#)).
- The Langevin bimolecular recombination model is used to model the recombination process of carriers and the generation process of singlet excitons (see [Bimolecular Recombination Model on page 538](#)).
- A singlet exciton equation is introduced to model the diffusion process, the generation from bimolecular recombination, the loss from decay, and the optical emissions of singlet excitons. Only Frenkel excitons (electron–hole pairs existing on the same molecule) participate in the optical process in organic materials (see [Singlet Exciton Equation on page 291](#)).

The organic device simulator is based on the work of Kozłowski [1], and other useful papers are available [2]–[10].

**Table 119** lists acronyms that describe organic device layers and transport mechanisms.

*Table 119      Commonly used acronyms for organic transport*

Acronym	Definition
EBL	Electron blocking layer
EML	Emission layer
ETL	Electron transport layer
HOMO	Highest occupied molecular orbital
HTL	Hole transport layer
LUMO	Lowest unoccupied molecular orbital
SCL	Space charge limited
TCL	Trapped charge limited
TSI	Thermally stimulated luminescence

---

## References

- [1] F. Kozłowski, *Numerical simulation and optimisation of organic light emitting diodes and photovoltaic cells*, PhD thesis, Technische Universität Dresden, Germany, 2005.
- [2] S.-H. Chang *et al.*, “Numerical simulation of optical and electronic properties for multilayer organic light-emitting diodes and its application in engineering education,” in *Proceedings of SPIE, Light-Emitting Diodes: Research, Manufacturing, and Applications X*, vol. 6134, pp. 26-1–26-10, 2006.
- [3] P. E. Burrows *et al.*, “Relationship between electroluminescence and current transport in organic heterojunction light-emitting devices,” *Journal of Applied Physics*, vol. 79, no. 10, pp. 7991–8006, 1996.
- [4] M. Hoffmann and Z. G. Soos, “Optical absorption spectra of the Holstein molecular crystal for weak and intermediate electronic coupling,” *Physical Review B*, vol. 66, no. 2, p. 024305, 2002.
- [5] J. Staudigel *et al.*, “A quantitative numerical model of multilayer vapor-deposited organic light emitting diodes,” *Journal of Applied Physics*, vol. 86, no. 7, pp. 3895–3910, 1999.
- [6] E. Tutiš *et al.*, “Numerical model for organic light-emitting diodes,” *Journal of Applied Physics*, vol. 89, no. 1, pp. 430–439, 2001.
- [7] B. Ruhstaller *et al.*, “Simulating Electronic and Optical Processes in Multilayer Organic Light-Emitting Devices,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 9, no. 3, pp. 723–731, 2003.
- [8] B. Ruhstaller *et al.*, “Transient and steady-state behavior of space charges in multilayer organic light-emitting diodes,” *Journal of Applied Physics*, vol. 89, no. 8, pp. 4575–4586, 2001.
- [9] A. B. Walker, A. Kambili, and S. J. Martin, “Electrical transport modelling in organic electroluminescent devices,” *Journal of Physics: Condensed Matter*, vol. 14, no. 42, pp. 9825–9876, 2002.
- [10] S. Odermatt, N. Ketter, and B. Witzigmann, “Luminescence and absorption analysis of undoped organic materials,” *Applied Physics Letters*, vol. 90, p. 221107, May 2007.

# 21

## Optical Generation

---

*This chapter describes various methods that are used to compute the optical generation rate when an optical wave penetrates into the device, is absorbed, and produces electron–hole pairs.*

The methods include simple optical beam absorption, the raytracing method, the transfer matrix method, the finite-difference time-domain, the beam propagation method, and loading external profiles from file. Different types of refractive index and absorption models, parameter ramping, and optical AC analysis are also described.

---

### Overview of Optical Generation

A unified interface for optical generation computation is available to provide a consistent simulation setup irrespective of the underlying optical solver methods. This allows for a gradual refinement of results and a balance of accuracy versus computation time in the course of a simulation, while only the solver-specific parameters have to change. Several approaches for computing the optical generation exist that are independent of the chosen optical solver:

- Compute the optical generation resulting from a monochromatic optical source.
- Compute the optical generation resulting from an illumination spectrum.
- Set a constant value for the optical generation rate either per region or globally.
- Read an optical generation profile from file.
- Compute the optical generation as a sum of the above contributions, possibly with different optical solvers or different solver-specific settings or excitation parameters.

For each of the contributions listed, a separate scaling factor can be specified. For transient simulations, it is possible to apply a time-dependent scaling factor that can be used, for example, to model the response to a light pulse or any other time-dependent light signal. This feature can also be used to improve convergence if the optical generation rate is very high. The unified interface also allows you to save the computed optical generation rate to a file for reuse in other simulations.

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

The optical generation resulting from a monochromatic optical source or an illumination spectrum is determined as the product of the absorbed photon density, which is computed by the optical solver, and the quantum yield. Several models for the quantum yield are available ranging from a constant scaling factor to a spatially varying quantum yield based on the band gap of the underlying material and the excitation wavelength of the optical source.

In the following sections, the different approaches for computing the optical generation and their corresponding parameters are described.

---

## Specifying the Type of Optical Generation Computation

In the `OpticalGeneration` section of the command file, at least one of the following methods to compute the optical generation must be specified; otherwise, the optical generation is not computed and is set to zero everywhere:

- `ComputeFromMonochromaticSource` activates optical generation computation with a single wavelength that is either specified in the `Excitation` section or as a ramping variable.
- `ComputeFromSpectrum` allows the sum of optical generation to be computed from an input spectrum of wavelengths.
- `ReadFromFile` imports the optical generation profile.
- `SetConstant` allows you to set a background constant optical generation in the specified region or material.

**Note:**

If you use the keyword `Scaling` in these optical generation methods, then the scaling applies only to quasistationary simulations. To scale transient simulations, see [Specifying Time Dependency for Transient Simulations on page 661](#).

If several methods are specified, then the total optical generation rate is given by the sum of the contributions computed with each method. The general syntax is:

```
Physics {
    ...
    Optics (
        OpticalGeneration (
            ...
            ComputeFromMonochromaticSource (...)
            ComputeFromSpectrum (...)
            ReadFromFile (...)
            SetConstant ( ... Value = <float> )
        )
        Excitation (...)
        OpticalSolver (...)
    )
}
```

## Chapter 21: Optical Generation

### Specifying the Type of Optical Generation Computation

```
        ComplexRefractiveIndex (...)
    )
}
```

Each method can have its own options such as a scaling factor or a particular time-dependency specification used in transient simulations. An example setup where the optical generation read from a file is scaled by a factor of 1.1 and a Gaussian time dependency is assumed for the monochromatic source is:

```
OpticalGeneration (
    ...
    ComputeFromMonochromaticSource (
        ...
        TimeDependence (
            WaveTime = (<t1>, <t2>)
            WaveTSigma = <float>
        )
    )
    ReadFromFile (
        ...
        Scaling = 1.1
    )
)
```

Both `Scaling` and `TimeDependence` can also be specified directly in the `OpticalGeneration` section if the same parameters will apply to all methods. If `TimeDependence` is specified directly in the `OpticalGeneration` section as well as in a source-specific section such as `ComputeFromMonochromaticSource`, then the latter takes precedence. For an overview of all available options, see [Table 253 on page 1644](#).

By default, the optical generation rate is calculated for every semiconductor region. However, you can suppress the computation of the optical generation rate for a specific region or material by specifying the keyword `-OpticalGeneration` in the corresponding region or material `Physics` section:

```
Physics ( region="coating" ) { Optics ( -OpticalGeneration ) }

Physics ( material="InP" ) { Optics ( -OpticalGeneration ) }
```

To visualize the optical intensity and generation, the keywords `OpticalIntensity` and `OpticalGeneration` must be added to the `Plot` section:

```
Plot {
    ...
    OpticalIntensity
    OpticalGeneration
}
```

Specifying `OpticalGeneration` in the `Plot` section plots not only the total optical generation that enters the electrical equations, but also the contributions from the different

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

methods of optical generation computation. See [Appendix F on page 1515](#) for the corresponding data names.

---

## Optical Generation From Monochromatic Source

Specifying `ComputeFromMonochromaticSource( . . . )` in the `OpticalGeneration` section activates the computation of the optical generation assuming a monochromatic light source. Details of the light source such as angle of incidence, wavelength, and intensity, and the optical solver used to model it must be set in the `Excitation` section and `OpticalSolver` section, respectively (see [Specifying the Optical Solver on page 669](#) and [Setting the Excitation Parameters on page 674](#)).

Together with the possibility of ramping parameters (see [Controlling Computation of Optical Problem in Solve Section on page 688](#)), for example, the wavelength of the incident light, this model can be used to simulate the optical generation rate as a function of wavelength.

---

## Illumination Spectrum

The optical generation resulting from a spectral illumination source, which is sometimes also known as *white light generation*, can be modeled in Sentaurus Device by superimposing the spectrally resolved generation rates. To this end, `ComputeFromSpectrum( . . . )` must be listed in the `OpticalGeneration` section. The illumination spectrum is then read from a text file whose name must be specified in the `File` section:

```
File {
    ...
    IlluminationSpectrum = "illumination_spectrum.txt"
}
Physics {
    ...
    Optics (
        ...
        OpticalGeneration (
            ...
            ComputeFromSpectrum ( . . . )
        )
    )
}
```

In its simplest form, the illumination spectrum file has a two-column format. The first column contains the wavelength in  $\mu\text{m}$  and the second column contains the intensity in  $\text{W}/\text{cm}^2$ . The characters # and \* mark the beginning of a comment.

As a default, the integrated generation rate resulting from the illumination spectrum is only computed once, that is, the first time the optical problem is solved and remains constant thereafter.

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

However, you can force its recomputation on every occasion by specifying the keyword `RefreshEveryTime` in the `ComputeFromSpectrum` section.

When the optical problem remains constant, that is, the absorbed photon density does not change, but the quantum yield is expected to vary (see [Quantum Yield Models on page 658](#)), specify the keyword `KeepSpectralData` in the `ComputeFromSpectrum` section. With this keyword, the spectral information of the last optical solution is kept in memory and only the quantum yield, which is needed to compute the optical generation rate and the optical absorption heat (see [Optical Absorption Heat on page 660](#)), is updated during the simulation.

Plotting spectral results requires the specification of a file name in the `File` section using the keyword `SpectralPlot` as well as the specification of the keyword `KeepSpectralData` in the `ComputeFromSpectrum` section. A file with extension `.tdr` is used to save spatial fields such as optical generation for each entry of the spectrum. Other results from the optical solver, for example, reflection, transmission, and absorption in the case of the TMM solver, as a function of the spectrum parameters are saved in a file with the extension `.plt`. The quantities saved in the `.plt` file are subject to the optical solver used in the simulation. To control the file name and when to plot, the same syntax in the `Solve` section applies to `SpectralPlot` as well as to `Plot`. See [When to Plot on page 178](#) and [Appendix G on page 1558](#).

In simulations that contain only a monochromatic source, the results from the optical solver are written to the current file. However, if a spectral source is present, the values should reflect the result of both the monochromatic source and the spectral source. Therefore, relative quantities (that is, unitless quantities) such as reflection are represented by a weighted average over all entries of the illumination spectrum and the monochromatic source if present. The weight is given by the corresponding number of incident photons. On the other hand, absolute quantities are represented by a simple sum over all spectral entries and the monochromatic source if present.

### Note:

The default for saving spectral plots is to write consecutive plots into one file. However, to write consecutive plots into several enumerated files, specify  
`SpectralPlot(-collected)=<string>`.

## Multidimensional Illumination Spectra

Often, illumination spectra depend on additional parameters, which might be related to an experimental setup that users want to model. For example, the intensity of the incident light might depend on not only the wavelength but also the angle of incidence. To account for such simulation setups, Sentaurus Device supports multidimensional illumination spectra. The format of a corresponding illumination spectrum file is:

```
# some comment
* another comment # and so on
```

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

```
Optics/Excitation/Wavelength [um] theta [deg] phi [deg] intensity
[W*cm^-2]
0.62 0 30 0.1
0.86 0 30 0.2
1.1 0 30 0.3
```

The header contains optional comment lines and a line defining the parameters assigned to each column. A parameter name or path is followed by its corresponding unit; if no unit is specified, the default unit of 1 is assumed. Listed parameters can refer either directly to existing parameters of the `Optics` section in Sentaurus Device or to user-defined parameters. The latter comes into play when using illumination spectra in combination with loading absorbed photon density or optical generation profiles from file. See [Loading Solution of Optical Problem From Files on page 749](#) for details.

**Note:**

The parameter `Intensity` is mandatory in every illumination spectrum file. However, the order of columns is arbitrary. Parameter names are case insensitive.

For multidimensional illumination spectrum files, the active columns must be selected in the command file. All other columns are ignored in the simulation. The required syntax in the `OpticalGeneration` section is:

```
ComputeFromSpectrum (
    ...
    Select (
        Parameter = ("Optics/Excitation/Wavelength" "Theta")
    )
)
```

The parameter `Intensity` is selected by default and does not need to be specified.

## Enhanced Spectrum Control

Being able to filter a given spectrum, based on a user-supplied condition, adds functionality to the computation of the optical generation resulting from a spectral illumination source. Specifying a static condition can be used to select a limited spectral range of interest or a subset of a multidimensional spectrum. The latter improves the handling of several spectra (for example, different standard spectra at various resolutions, and measured spectra) since they can be compiled in a single file and still be addressed separately.

A condition is called *dynamic* if it changes during the simulation. For example, a dynamic condition can include the excitation wavelength, which is ramped in a `Quasistationary` statement. Dynamic conditions can be used to ramp through different spectra or to superimpose a fixed spectrum with a varying spectrum.

## Chapter 21: Optical Generation

### Specifying the Type of Optical Generation Computation

To specify a condition, a Tcl-compatible expression enclosed in double quotation marks must be provided in the `Select` section:

```
ComputeFromSpectrum (
    ...
    Select (
        Parameter = ("Wavelength" "Theta")
        Condition = "$wavelength > 0.3 && $wavelength < 1.2"
    )
)
```

Identifiers preceded by a dollar sign (\$) such as `wavelength` in the above example are considered to be variables referring to the respective column in the illumination spectrum file. Sentaurus Device extracts a subset of the spectrum by applying the condition expression to each row of the spectrum defined in the file. Before the expression is passed to the global Tcl interpreter of Sentaurus Device for evaluation, variables are substituted with their corresponding row-specific values. If the expression evaluates to true, the row is considered to be an active entry of the spectrum used in the `ComputeFromSpectrum` computation; otherwise, it is ignored.

#### Note:

By default, duplicate entries of the spectrum are ignored. However, specifying the keyword `AllowDuplicates` in the `Select` section will retain such entries.

Identifiers such as Sentaurus Device parameter names and variable names referring to the respective column in the illumination spectrum file are treated as case insensitive in the `Parameter` list and the `Condition` statement of the `Select` section.

Assuming a spectrum file containing different spectra distinguished by their names of the form:

```
wavelength [um] intensity [W*cm^-2] spectrum [1]
0.2 0.0012 "AM1.5g"
0.3 0.0034 "AM1.5g"
...
0.2 0.0056 "AM0"
...
```

a single spectrum can be selected by specifying the following in the `ComputeFromSpectrum` section:

```
Select (
    Parameter = ("Wavelength" "Spectrum")
    Condition = "$Spectrum == \\"AM1.5g\\" "
)
```

#### Note:

Double quotation marks in the Tcl expression must be escaped to avoid conflicts with the parser of the Sentaurus Device command file.

## Chapter 21: Optical Generation

### Specifying the Type of Optical Generation Computation

For a condition to change during the simulation, it must reference an internal variable of Sentaurus Device that is ramped in a Quasistationary statement. Sentaurus Device interprets identifiers without a preceding \$ in the condition expression as internal parameters. If necessary due to ambiguity, internal parameters also can be specified using their full path such as Optics/Excitation/Wavelength. Specifying an identifier that cannot be matched with an internal parameter results in an error.

To select a subspectrum based on a dynamic condition, assume the following spectrum file:

```
wavelength [um] intensity [W*cm^-2] centralWavelength [nm]
0.2 0.0012 300
0.3 0.0034 300
...
0.3 0.0056 400
0.4 0.0068 400
...
```

The following command file syntax shows how to ramp through the various spectra identified by their central wavelength:

```
Physics {
    Optics (
        OpticalGeneration (
            ComputeFromSpectrum (
                Select (
                    Parameter= ("Wavelength" "centralWavelength")
                    Condition= "abs(Wavelength - $centralWavelength*1e-3)
                                < 1e-6"
                )
            )
        )
    )
}
Solve {
    Quasistationary (
        Goal { modelParameter = "Wavelength" value = 1.2 }
    ) { Coupled {Poisson Electron Hole} }
}
```

#### Note:

In the condition expression, users are responsible for rescaling the spectrum variables, if necessary, when comparing them to internal parameters having a fixed unit. Despite the fact that the units of the spectrum variables are known to Sentaurus Device, it is unclear whether they are related to internal parameters as the names of spectrum variables are arbitrary.

When comparing floating-point numbers for equality in a condition expression, it is recommended that you check that the absolute value of their difference is smaller than a user-specified epsilon as shown in the examples. This avoids any unexpected precision issues resulting from numeric operations or reading floating-point numbers from file.

## Chapter 21: Optical Generation

### Specifying the Type of Optical Generation Computation

For more flexibility, the Select section contains an auxiliary keyword `Var`, which holds a floating-point value. It has no impact on the optical generation computation as such, but it can be ramped in a Quasistationary statement like any other internal parameter that supports ramping. Therefore, the keyword `Var` allows you to create dynamic conditions without affecting the results of any other optical generation contributions defined in the OpticalGeneration section such as `ComputeFromMonochromaticSource`. This is demonstrated by the following syntax, which is based on the above example where Sentaurus Device ramps through various spectra, but it contains a fixed monochromatic source:

```
Physics {
    Optics (
        Excitation (
            Wavelength = 0.6
        )
        OpticalSolver ( ... )
        OpticalGeneration (
            ComputeFromMonochromaticSource ( ... )
            ComputeFromSpectrum (
                Select (
                    Parameter= ("Wavelength" "centralWavelength")
                    Condition= "abs(Var - $centralWavelength*1e-3) < 1e-6"
                )
            )
        )
    )
}
Solve {
    Quasistationary (
        Goal { modelParameter = "Var" value = 1.2 }
    ) { Coupled {Poisson Electron Hole} }
}
```

The keyword `Var` can be plotted by specifying `ModelParameter="Physics/Optics/OpticalGeneration/ComputeFromSpectrum>Select/Var"` in the CurrentPlot section of the command file.

---

## Loading and Saving Optical Generation From and to Files

Sometimes, solving the optical problem might require long computation times, in which case, it can be useful to save the solution to a file and to load the optical generation or absorbed photon density profile in later simulations whose optical properties remain constant. In the following example, optical generation is used as a synonym for absorbed photon density. The command file syntax for saving the optical generation rate to a file is:

```
File {
    OpticalGenerationOutput = <filename>
}
```

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

For loading the optical generation rate from a file, the syntax is:

```
File {
    OpticalGenerationInput = <filename>
}

Physics {
    Optics (
        OpticalGeneration (
            ReadFromFile (
                DatasetName = AbsorbedPhotonDensity | OpticalGeneration
            )
        )
    )
}
```

If the input file to be loaded contains both an optical generation and an absorbed photon density profile, the keyword `DatasetName` controls which one to use. By default, the absorbed photon density is used. The optical generation profile to be loaded also can be defined on a mixed-element grid that is different from the one used in the device simulation, or on a tensor grid resulting from an EMW simulation. In that case, the profile is interpolated automatically onto the simulation grid upon loading. For more details on how to control the interpolation, including the truncation and shifting of the interpolation domain, see [Controlling Interpolation When Loading Optical Generation Profiles on page 768](#).

Further options of the `ReadFromFile` section can be found in [Table 253 on page 1644](#). Similar but more powerful functionality is provided by the feature `FromFile` (see [Loading Solution of Optical Problem From Files on page 749](#)).

---

## Constant Optical Generation

Assigning a constant optical generation rate to a certain region or material is achieved by specifying a value for a particular region or material as shown here:

```
Physics (Region = <region name>) {
    ...
    Optics (
        OpticalGeneration (
            SetConstant (Value = <float>)
        )
    )
}
Physics (Material = <material name>) {
    ...
    Optics (
        OpticalGeneration (
            SetConstant (Value = <float>)
        )
    )
}
```

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

If all semiconductor regions are supposed to have the same optical generation rate, its value is best specified in the global `Physics` section as follows:

```
Physics {
    ...
    Optics (
        OpticalGeneration (
            SetConstant (Value = <float>)
        )
    )
}
```

**Note:**

The constant optical generation model is functionally equivalent to the constant carrier generation model presented in [Constant Carrier Generation Model on page 494](#).

---

## Quantum Yield Models

The quantum yield model describes how many of the absorbed photons are converted to generated electron–hole pairs. The simplest model, `QuantumYield(Unity)`, assumes that all absorbed photons result in generated charge carriers irrespective of the band gap or other properties of the underlying material. This corresponds to a global quantum yield factor of one, which is the default value, except for nonsemiconductor regions where it is always set to zero even if photons are actually absorbed.

A more realistic model, `QuantumYield(Stepfunction(...))`, takes the band gap into account. If the excitation energy is greater than or equal to the bandgap energy  $E_g$ , the quantum yield is set to one; otherwise, it is set to zero. The bandgap energy used can be set directly by specifying a value for `Energy` in eV or, alternatively, a corresponding `Wavelength` in micrometers. Another option is `Bandgap`, which uses the temperature-dependent bandgap energy as given in [Band Gap and Electron Affinity on page 305 \(Equation 153\)](#). To include bandgap narrowing effects in the form of [Equation 158](#), the keyword `EffectiveBandgap` must be specified.

In the presence of free carrier absorption (FCA), which is activated in Sentaurus Device by specifying `CarrierDep(Imag)` in the `ComplexRefractiveIndex` section of the command file, the spatially varying quantum yield factor is reduced according to the ratio of free carrier absorption,  $\alpha_{\text{FCA}}$ , to total absorption,  $\alpha_{\text{tot}}$ :

$$\eta_G = \eta_{G_0} \left(1 - \frac{\alpha_{\text{FCA}}}{\alpha_{\text{tot}}}\right) \quad (689)$$

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

where  $\alpha_{\text{FCA}}$  and  $\alpha_{\text{tot}}$  are determined by the `ComplexRefractiveIndex` specification in the parameter file, given the following relation between the absorption coefficient and the extinction coefficient:

$$\alpha = \frac{4\pi k}{\lambda} \quad (690)$$

The change of the extinction coefficient due to free carrier absorption is represented by  $\Delta k_{\text{carr}}$  (see [Carrier Dependency on page 695](#)). The prefactor  $\eta_{G_e}$  can be set to 1 by specifying the keyword `EffectiveAbsorption` in the `QuantumYield` section, or it can represent a step function: If the photon energy is sufficiently large to allow for interband optical absorption, it is 1, or otherwise 0. The latter requires the specification of a `Stepfunction` section, which takes precedence over `EffectiveAbsorption` if both are specified. The command file syntax for specifying quantum yield models is:

```
Physics {
    Optics (
        OpticalGeneration (
            ...
            QuantumYield = <float>
            QuantumYield ( Unity )
            QuantumYield (
                EffectiveAbsorption
                StepFunction (
                    Wavelength = <float> #[um]
                    # OR
                    Energy = <float>      #[eV]
                )
                StepFunction ( Bandgap | EffectiveBandgap )
            )
        )
    )
}
```

For more sophisticated quantum yield models, Sentaurus Device offers a PMI interface (see [Optical Quantum Yield on page 1367](#)). All quantum yield models also can be specified in a region or material `Physics` section. The resulting quantum yield can be plotted by specifying `QuantumYield` in the `Plot` section.

### Note:

By default, the optical absorption due to `ComputeFromSpectrum` is computed only once; any further changes in the quantum yield are neglected. To account for varying quantum yield, it is necessary to recompute the corresponding optical absorption by using `RefreshEveryTime` in the `ComputeFromSpectrum` section. However, this will impact simulator performance depending on the size of the spectrum and the chosen optical solver.

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

### Optical Absorption Heat

The absorbed photon energy  $E_{ph}$  is distributed among different processes by quantum yield factors. The following two channels are accounted for:

- Thermalization to the band gap (interband absorption): When a photon is absorbed across the band gap in a semiconductor, it is absorbed to create an electron–hole pair. The excess energy (photon energy minus the band gap) of the new electron–hole pair is assumed to thermalize, resulting eventually in lattice heating.
- Complete thermalization (intraband absorption): In the case of the photon energy being smaller than the band gap, the photon can be absorbed to increase the energy of a carrier. The excess energy relaxes eventually, contributing to lattice heating.

In both processes, it is assumed that the eventual lattice heating occurs in the locality of photon absorption. The corresponding energy equation reads:

$$E_{ph} = \eta_{T_{Eg}}(E_{ph} - E_g) + \eta_G E_g + \eta_{T_0} E_{ph} \quad (691)$$

where the energy contributions of the first term and the third term are dissipated into the lattice through thermalization. The energy of the second term is consumed for the generation of an electron–hole pair.

In general,  $\eta_{T_{Eg}} = \eta_G$  since it is assumed that every photon generates a charge carrier, while the residual energy  $E_{ph} - E_g$  is dissipated into the lattice. Therefore, ignoring free carrier absorption, if the chosen quantum yield model evaluates to 1,  $\eta_{T_{Eg}} = \eta_G = 1$  and  $\eta_{T_0} = 0$ . If the quantum yield model evaluates to 0,  $\eta_{T_{Eg}} = \eta_G = 0$  and  $\eta_{T_0} = 1$ .

Distinguishing between  $\eta_{T_{Eg}}$  and  $\eta_G$  allows for the control of multiple-generation processes as well. For example, in the UV spectrum, it is possible that  $E_{ph} > 2E_g$ , and so more than one charge carrier can be generated per absorbed photon. To model multiple-generation processes, the `OpticalQuantumYield` PMI (see [Optical Quantum Yield on page 1367](#)) must be used where all quantum yield factors can be specified independently for each vertex.

Supporting several optical absorption processes affects the value of  $\eta_G$  as can be seen from [Equation 691](#). It no longer depends on the local effective bandgap energy only. Factoring in free carrier absorption means that photons absorbed through this process do not contribute to the optical generation rate, which is used in the drift-diffusion equations. In general, the quantum yield factors are independent as long as the energy equation is fulfilled locally.

The quantum yield factor  $\eta_G$  is given by [Equation 689](#), and the quantum yield factor attributed to complete thermalization is computed as:

$$\eta_{T_0} = 1 - \eta_G \quad (692)$$

The quantum yield factor  $\eta_{T_{Eg}}$  is set to  $\eta_G$  unless it is specified explicitly using the `OpticalQuantumYield` PMI.

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

### Note:

If  $\eta_G$  is set to a constant value in the command file, then  $\eta_{T_0}$  is still given by [Equation 692](#) to ensure energy balance.

Specifying `OpticalAbsorptionHeat` and `ThermalizationYield` in the `Plot` section of the command file results in the following quantities being written to the plot file for each vertex:

- `OpticalAbsorptionHeat`:  $(\eta_{T_{Eg}}(E_{ph} - E_g) + \eta_{T_0}E_{ph})N_{ph}$
- `OpticalAbsorptionHeat_Bandgap`:  $\eta_{T_{Eg}}(E_{ph} - E_g)N_{ph}$
- `OpticalAbsorptionHeat_Vacuum`:  $\eta_{T_0}E_{ph}N_{ph}$
- `ThermalizationYield_Bandgap`:  $\eta_{T_{Eg}}$
- `ThermalizationYield_Vacuum`:  $\eta_{T_0}$

Here,  $N_{ph}$  represents the number of absorbed photons.

### Note:

`OpticalAbsorptionHeat` is not calculated for optical absorption that is loaded using `ReadFromFile` or for constant optical generation specified by `SetConstant` because the corresponding wavelength of the light source is unknown.

---

## Specifying Time Dependency for Transient Simulations

To model the electrical response of a light pulse, incident on a device, a description of the light signal over time can be specified either globally or separately for each type of optical generation computation. For the former, `TimeDependence( . . . )` is listed directly in the `OpticalGeneration` section, while for the latter, it is given as an argument of the chosen type of optical generation computation.

If you specify `TimeDependence` both globally and for a specific type of optical generation computation such as `ComputeFromMonochromaticSource`, the latter takes precedence. `TimeDependence` can only be specified in the global `Physics` section and not for a particular region or material.

The given time dependency essentially scales the optical generation rate, resulting from a stationary solution of the optical problem as a function of time. The time dependency is only taken into account inside a `Transient` statement and the corresponding scaling factor for the different types of optical generation computation is automatically written to the `Current` file under the name `TimeDependence(Ft)`. If the optical generation needs to be scaled inside a `Quasistationary`, the keyword `Scaling` in the `OpticalGeneration` section can be used. However, this scaling factor does not apply inside a `Transient` statement. Instead, `Scaling` can be set directly in the respective `TimeDependence` section.

## Chapter 21: Optical Generation

### Specifying the Type of Optical Generation Computation

Different types of time dependency are available:

- Linear time dependency
- Gaussian time dependency
- Exponential time dependency
- Cosine time dependency
- Arbitrary time dependency read from a file

**Note:**

If no time dependency has been specified, a scaling factor of 1 is used inside a `Transient` statement irrespective of any other scaling factors set in the `OpticalGeneration` section.

In addition, each of the above time dependencies can be extended to a periodic signal of the respective type. For the analytic time dependencies, you can specify a time interval `WaveTime= (<t1>, <t2>)`. Before `<t1>`, the optical generation rate undergoes an increase from zero characterized by the type of analytic time dependency. After `<t2>`, the optical generation rate experiences a corresponding decrease.

By default, Sentaurus Device adds turning points (see [Time-Stepping on page 140](#)) to the `Transient` statement at several characteristic time points of the light pulse to help convergence and to ensure it is resolved even if the time step of the `Transient` statement is larger than the entire pulse width. For details about optical turning points, see [Optical Turning Points on page 667](#).

The linear time function as shown in [Figure 25](#) is expressed as:

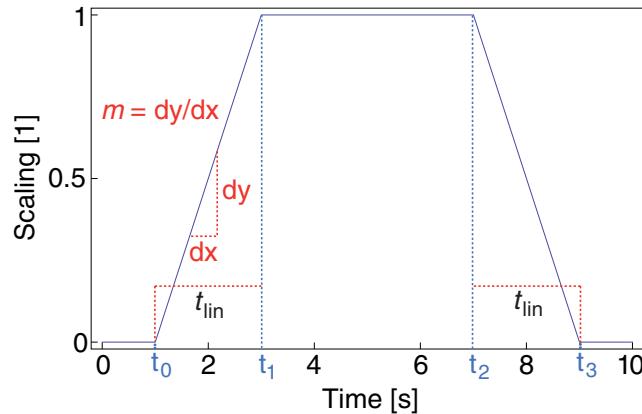
$$F(t) = \begin{cases} \overset{\circ}{\rightarrow} \max(0, m(t - t_1) + 1) & , t < t_1 \\ \overset{\circ}{\rightarrow} 1 & , t_1 \leq t \leq t_2 \\ \overset{\circ}{\rightarrow} \max(0, m(t_2 - t) + 1) & , t > t_2 \end{cases} \quad (693)$$

where  $m$  is given by `WaveTSlope`. Alternatively, `WaveTLin` can be specified, which is the inverse of  $m$ , that is, it corresponds to the rise time  $t_{lin} = t_1 - t_0$  in seconds.

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

Figure 25 Rise and decay times based on a linear function

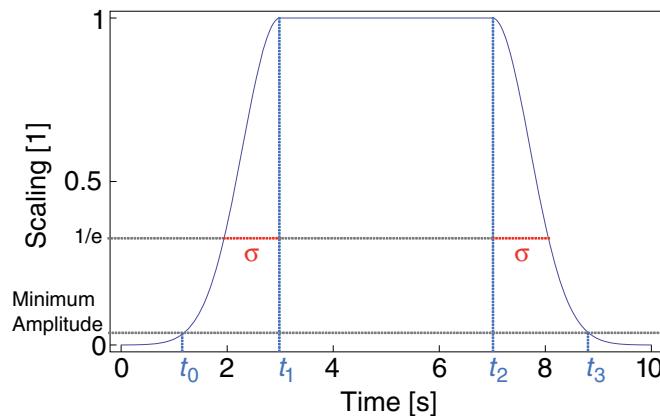


The Gaussian time function as shown in Figure 26 is expressed as:

$$F(t) = \begin{cases} \exp -\frac{(t_1-t)^2}{\sigma^2}, & t < t_1 \\ 1, & t_1 \leq t \leq t_2 \\ \exp -\frac{(t-t_2)^2}{\sigma^2}, & t > t_2 \end{cases} \quad (694)$$

where  $\sigma$  is defined by WaveTSigma.

Figure 26 Rise and decay times based on a Gaussian function



## Chapter 21: Optical Generation

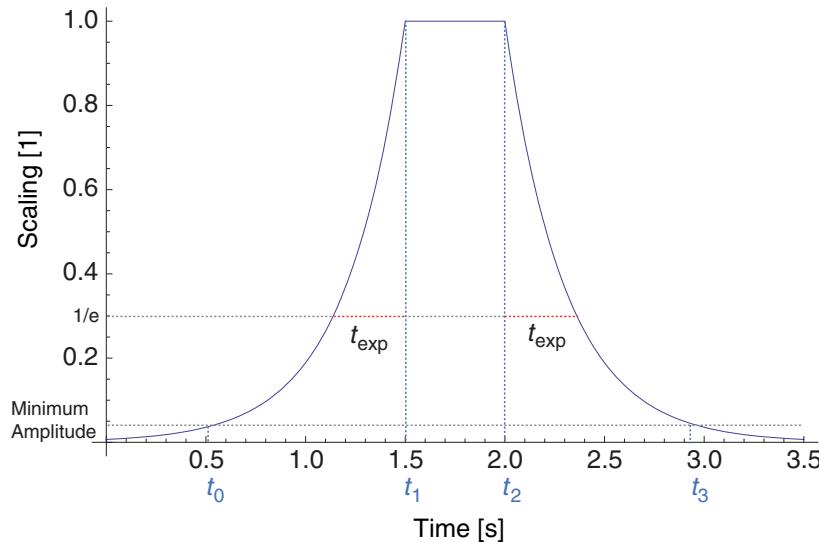
### Specifying the Type of Optical Generation Computation

The exponential time function as shown in [Figure 27](#) is expressed as:

$$F(t) = \begin{cases} \exp\left(\frac{t-t_1}{t_{\exp}}\right) & t < t_1 \\ 1 & t_1 \leq t \leq t_2 \\ \exp\left(\frac{t_2-t}{t_{\exp}}\right) & t > t_2 \end{cases} \quad (695)$$

where  $t_{\exp}$  is defined by `WaveTExp`.

*Figure 27 Rise and decay times based on an exponential function*



The cosine time function as shown in [Figure 28](#) is expressed as:

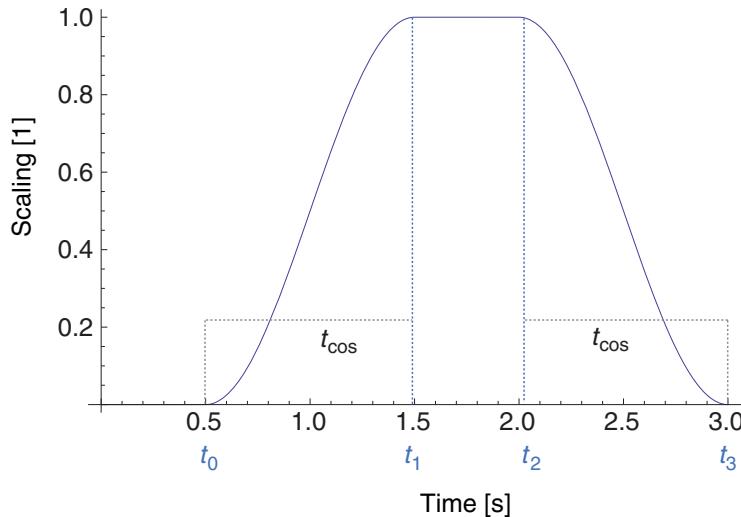
$$F(t) = \begin{cases} 0 & t \leq t_1 - t_{\cos} \vee t \geq t_2 + t_{\cos} \\ \frac{1}{2} \cdot 1 + \cos\left(\frac{\pi(t-t_1)}{t_{\cos}}\right) & t_1 - t_{\cos} < t < t_1 \\ 1 & t_1 \leq t \leq t_2 \\ \frac{1}{2} \cdot 1 + \cos\left(\frac{\pi(t-t_2)}{t_{\cos}}\right) & t_2 < t < t_2 + t_{\cos} \end{cases} \quad (696)$$

where  $t_{\cos}$  is defined by `WaveTCos`.

## Chapter 21: Optical Generation

### Specifying the Type of Optical Generation Computation

*Figure 28 Rise and decay times based on a cosine function*



If no time interval is specified,  $t_1 = t_2 = 0$  is assumed. The analytic time dependencies are selected by specifying a value for the corresponding parameters as shown in [Table 120](#).

*Table 120 Selection of analytic time dependency*

Time dependency	Required keyword
Linear	WaveTSlope or WaveTLin
Gaussian	WaveTSigma
Exponential	WaveTExp
Cosine	WaveTCos

An arbitrary time dependency can be applied by defining a time function whose interpolation points are given in a file with a white space-separated, two-column format. The first column contains the time points in seconds and the second column contains the corresponding function values. Linear interpolation is used to obtain function values between the interpolation points.

This type of time dependency can be activated using the following syntax:

```
File {
    OptGenTransientScaling= <filename>      # file containing interpolation
                                                # points
}
Physics {
    Optics (
```

## Chapter 21: Optical Generation

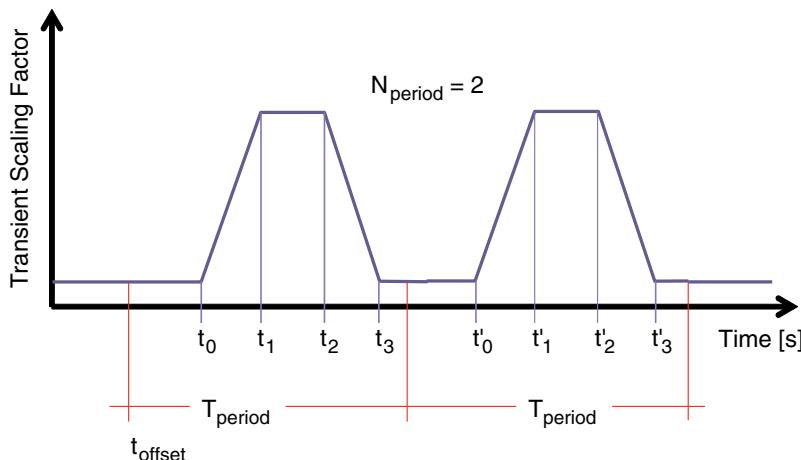
### Specifying the Type of Optical Generation Computation

```
OpticalGeneration (
    TimeDependence (FromFile)
)
}
```

To transform any of the above time dependencies into a periodic signal, its period in seconds must be specified using the keyword `WaveTPeriod`. The specified signal is repeated infinitely, unless the number of periods is set explicitly with the keyword `WavePeriods`.

For linear and Gaussian time dependency, an additional temporal offset can be defined to essentially control the relative location of the signal within the period as illustrated in [Figure 29](#).

*Figure 29 Extension of a predefined time dependency to a periodic signal where  $T_{period}$ ,  $N_{period}$ , and  $t_{offset}$  correspond to `WaveTPeriod`, `WavePeriods`, and `WaveTPeriodOffset`*



The syntax for the linear periodic signal shown in [Figure 29](#) is:

```
Physics {
    Optics (
        OpticalGeneration (
            TimeDependence (
                WaveTime = (t1, t2)
                WaveTSlope = m
                WaveTPeriod = Tperiod
                WavePeriods = Nperiod
                WaveTPeriodOffset = toffset
            )
        )
    )
}
```

## Chapter 21: Optical Generation

Specifying the Type of Optical Generation Computation

### Optical Turning Points

Typically, you specify turning points as described in [Time-Stepping on page 140](#) in the Transient statement to limit the advancing time step. Controlling time-stepping based on knowledge about the device physics and its characteristics can improve the convergence behavior. Light pulses in transient simulations have a strong impact on device characteristics and, therefore, corresponding optical turning points are added by default. The default optical turning points also guarantee that a light pulse is resolved to a certain degree, independent of the advancing time step before the start time of the pulse. You can switch off the default optical turning points or change their parameters in the command file.

In contrast to general turning points described in [Time-Stepping on page 140](#), optical turning points are specified in the `OpticalTurningPoints` section within the `TimeDependence` section with the definition of the light pulse.

The predefined optical turning points correspond to the time points  $t_0$ ,  $t_1$ ,  $t_2$ , and  $t_3$  shown in [Figure 25](#) to [Figure 29](#), for which the advancing time step can be limited using the keywords `DtRiseStart`, `DtRiseEnd`, `DtFallStart`, and `DtFallEnd`, respectively. In addition, the time step can be limited for the ranges  $[t_0 t_1]$ ,  $[t_1 t_2]$ , and  $[t_2 t_3]$  that are associated with the keywords `DtRise`, `DtPlateau`, and `DtFall`. You can limit the maximum time step in these ranges separately using the aforementioned keywords or set the same maximum time step for all three time ranges using the keyword `Dt`. If both `Dt` and a range-specific keyword such as `DtPlateau` are specified, then the value of the range-specific keyword is used.

#### Note:

Limiting the maximum time step within a range, using `DtRise`, `DtPlateau`, or `DtFall`, has an effect only if the time-stepping is such that the calculated time step used to advance from the last time point  $t_i$  before a range is not larger than  $t_3 - t_i$ . In other words, limiting the time step becomes active only if the regular time-stepping enters the predefined range. See [Time-Stepping on page 140](#) for details about the behavior of turning points.

Specifying a time step equal to 0 switches off the corresponding optical turning point or range.

#### Note:

The points  $t_0 - t_3$  are enabled by default; whereas, the ranges must be enabled explicitly by specifying a limiting time step greater than 0. To switch off all optical turning points and ranges, specify `-OpticalTurningPoints` in the respective `TimeDependence` section.

For analytic light signals with asymptotically decaying tails such as a Gaussian, the time points  $t_0$  and  $t_3$  are not naturally defined. By default, they are chosen to be the time points where the signal amplitude is equal to 0.01.

## Chapter 21: Optical Generation

### Solving the Optical Problem

However, by setting `MinAmplitude` in the `OpticalTurningPoints` section to a different value, you can shift the time points  $t_0$  and  $t_3$  in either direction.

For time-dependence `FromFile`, only the time points  $t_0$  and  $t_3$  and a single range  $[t_0 t_3]$  are supported, which are active by default. The limiting time step for the range can be set with the keyword `Dt`. The time points  $t_0$  and  $t_3$  are associated with the first and last time points given in the `OptGenTransientScaling` file specified in the `File` section.

The following example demonstrates the syntax for optical turning points (an overview of the various parameters and corresponding default values is given in [Table 254 on page 1646](#)):

```
Physics {
    Optics (
        OpticalGeneration (
            TimeDependence (
                WaveTime = (t1, t2)
                WaveTSigma = σ
                OpticalTurningPoints (
                    DtRiseStart = 1e-4      # [s]
                    DtFallEnd = 0          # switch off turning point t3
                    MinAmplitude = 0.005
                    DtPlateau = 1e-3       # [s]
                    DtFall = 2e-4          # [s]
                )
            )
        )
    )
}
```

---

## Solving the Optical Problem

`ComputeFromMonochromaticSource` and `ComputeFromSpectrum` require the solution of the optical problem for a given excitation to obtain the optical generation rate. Several optical solvers are available and the choice for a specific method is determined usually by the optimum combination of accuracy of results and computation time.

Besides selecting a certain optical solver and specifying its particular parameters, it is necessary to define the excitation parameters, to choose an appropriate refractive index model, and to control when a solution is computed in the `Solve` section. These steps are explained in the following sections.

## Specifying the Optical Solver

The following optical solvers are supported:

- Transfer matrix method (TMM)
- Finite-difference time-domain (FDTD)
- Raytracing (RT)
- Beam propagation method (BPM)
- Loading solution of optical problem from file
- Optical beam absorption method
- Composite method

**Note:**

By default, the log of the optical solver is written to standard output and the simulation log file as part of the general simulation log. However, depending on the specific setup, the optical solver log can be verbose and affect the overall readability of the simulation log. To avoid this behavior, you can either:

- Set `Verbosity=0` in the `Optics` section to completely suppress any logging activity of the optical solver except for warning and error messages.
- Redirect the log of the optical solver to a separate log file by specifying the keyword `OpticsOutput` in the global `File` section.

## Transfer Matrix Method

The TMM solver is selected using the following syntax:

```
Physics {
    ...
    Optics (
        ...
        OpticalSolver (
            TMM ( <TMM_options> )
        )
    )
}
```

The TMM-specific options, such as the parameters for the extraction of the layer stack, are described in [Using Transfer Matrix Method on page 739](#).

## Chapter 21: Optical Generation

Solving the Optical Problem

### Finite-Difference Time-Domain Method

In contrast to the TMM solver, the FDTD-specific parameters, such as boundary conditions and extractors, are defined in a separate command file that is set in the `File` section with the keyword `OpticalSolverInput`. To provide some basic control of the FDTD solver from within Sentaurus Device, excitation parameters such as `Wavelength`, `Theta`, and `Phi`, as well as the polarization `Psi`, are overwritten by their counterparts in the `Excitation` section of the Sentaurus Device command file if specified.

#### Note:

The `Psi` keyword in EMW corresponds to the `PolarizationAngle` keyword in Sentaurus Device.

The FDTD solver in Sentaurus Device supports both the 2D and 3D excitation specification of EMW as outlined in *Sentaurus™ Device Electromagnetic Wave Solver User Guide*, Specifying Direction and Polarization.

The syntax for activating the FDTD solver requires as input the name of the command file of the FDTD solver. The general syntax is:

```
File {  
    OpticalSolverInput= <EMW_command_file>  
}  
Physics {  
    Optics {  
        OpticalSolver (  
            FDTD ( ... )  
        )  
    }  
}
```

The FDTD algorithm is based on a tensor mesh, which can be generated independently before the device simulation. Another option is to build the tensor mesh during the simulation before the call to EMW. The main advantage of the latter option is that the tensor grid can be adjusted to a possibly changing excitation wavelength in a `Quasistationary` statement. Since the accuracy and stability of the FDTD method crucially depend on the discretization with respect to the wavelength, this feature becomes important when a large range of the light spectrum is scanned (see [Illumination Spectrum on page 651](#) and [Parameter Ramping on page 689](#)).

To activate this feature, `GenerateMesh( . . . )` must be specified in the `FDTD` section and a common base name (that is, file name excluding suffix) for the boundary file and the Sentaurus Mesh command file must be defined in the `File` section using the keyword `MesherInput`. Sentaurus Mesh is then called with the specified base name prepended by the basename of the `Plot` file given in the `File` section of the Sentaurus Device command file.

The resulting tensor grid file is detected automatically and replaces the grid file in the user-provided EMW command file.

## Chapter 21: Optical Generation

### Solving the Optical Problem

By default, a tensor mesh is generated before the first call to EMW and remains in use during further calls to the solver. However, if the excitation wavelength varies and the initially built tensor mesh no longer fulfills the requirements, two options exist that control the update of the mesh.

Using the keyword `ForEachWavelength` in the `GenerateMesh` section triggers the computation of a new tensor mesh whenever the wavelength changes compared to the previous solution of the FDTD solver. Internally, the wavelength specified in the user-provided Sentaurus Mesh command file is replaced with the current value.

Since the slope of the complex refractive index as a function of wavelength can vary from almost zero to high values, depending on the wavelength interval, it is possible to limit the mesh generation according to a list of strictly monotonically increasing wavelengths. If the current wavelength enters a new interval, the mesh is updated and remains in use until the wavelength moves beyond the interval boundaries. The syntax for such a use case is:

```
FDTD (
    ...
    GenerateMesh (
        Wavelength = ( 0.35 0.55 0.7 0.8 )      # wavelength in μm
    )
)
```

For the command file syntax and options of the FDTD solver, see the *Sentaurus™ Device Electromagnetic Wave Solver User Guide*.

## Raytracing

More details about the raytracer can be found in [Raytracing on page 702](#). The raytracer requires the use of the complex refractive index model, and various excitation variables can be ramped. These rampable excitation variables are `Intensity`, `Wavelength`, `Theta`, `Phi`, and `PolarizationAngle` or `Polarization`.

The required syntax is:

```
Physics {...}
    Optics (
        ComplexRefractiveIndex(...)
        OpticalGeneration(...)
        OpticalSolver(
            RayTracing(...)
        )
        Excitation(...)
    )
}
```

## Chapter 21: Optical Generation

### Solving the Optical Problem

where the RayTracing section contains:

```
RayTracing(
    # Setting keywords of raytracer
    # -----
    CompactMemoryOption
    MonteCarlo
    OmitReflectedRays
    OmitWeakerRays
    RedistributeStoppedRays

    PolarizationVector = vector
    PolarizationVector = Random
    DepthLimit = integer
    MinIntensity = float           # fraction, relative value
    RetraceCRIchange = float       # fractional change to retrace rays
    VirtualRegions { "string" "string" ... }
    VirtualRegions {}
    ExternalMaterialCRIFile = "string"
    WeightedOpticalGeneration

    # Defining starting rays:
    # -----
    RayDistribution( ... )
    RectangularWindow (
        RectangleV1 = vector      # vertex 1 of rectangular window [um]
        RectangleV2 = vector      # vertex 2 of rectangular window [um]
        RectangleV3 = vector      # vertex 3 needed only for 3D [um]
        # number of rays = LengthDiscretization x WidthDiscretization
        LengthDiscretization = integer # longer side
        WidthDiscretization = integer # shorter side needed only for 3D
        RayDirection = vector
    )
    UserWindow (
        NumberOfRays = integer     # number of rays in file
        RaysFromFile = "filename.txt" # position(um) direction area (cm^2)
        PolarizationVector = ReadFromExcitation
    )
)
```

Some comments about the RT syntax:

- The keyword `RetraceCRIchange` specifies the fractional change of the complex refractive index (either the real or imaginary part) from its previous state that will force a total recomputation of raytracing.
- Starting rays for raytracing can be set using `RectangularWindow` or `UserWindow`. Details can be found in [Window of Starting Rays on page 709](#). Only one of the windows can be chosen but not both.

## Chapter 21: Optical Generation

### Solving the Optical Problem

- Starting rays also can be set using the illumination window (see [Illumination Window on page 676](#)) and the `RayDistribution` section (see [Distribution Window of Rays on page 711](#)).
- Raytracing in cylindrical coordinates is also possible (see [Cylindrical Coordinates for Raytracing on page 712](#)).

## Beam Propagation Method

The BPM solver is selected using the following syntax:

```
Physics {
    Optics (
        OpticalSolver (
            BPM (<BPM_options>)
        )
    )
}
```

The BPM-specific options, such as the specific excitation type or the discretization parameters, are described in [Using Beam Propagation Method on page 758](#).

## Loading Solution of Optical Problem From File

The solution of the optical problem, which can be either an absorbed photon density profile or an optical generation profile, also can be loaded from a file. In contrast to using `OpticalGeneration ( ReadFromFile ( ... ) )`, the optical solver `FromFile` offers more flexibility for simulation setups that involve an illumination spectrum or parameter ramping.

The required syntax is:

```
File {
    OpticalSolverInput = "<file name or file name pattern>"
}
Physics {
    Optics (
        OpticalGeneration (
            ComputeFromMonochromaticSource ( )
        )
        OpticalSolver (
            FromFile (<FromFile_options>)
        )
    )
}
```

The use of the optical solver `FromFile` is described in [Loading Solution of Optical Problem From Files on page 749](#).

## Chapter 21: Optical Generation

Solving the Optical Problem

### Optical Beam Absorption Method

The following syntax is used to select the optical beam absorption method as the optical solver:

```
Physics {  
    Optics (  
        OpticalSolver (  
            OptBeam (<OptBeam_options>)  
        )  
    )  
}
```

The OptBeam-specific options, such as the parameters for the extraction of the layer stack, are described in [Using Optical Beam Absorption Method on page 755](#).

### Composite Method

The following syntax is used to select the composite method as the optical solver:

```
Physics {  
    Optics (  
        OpticalSolver (  
            Composite (<Composite_options>)  
        )  
    )  
}
```

The Composite solver itself is not a numeric method to solve an optical problem. Instead, it coordinates the execution of other solvers that might depend on each other and sums their results to yield a general solution. As such, it requires the specification of at least another optical device instance. For details about this solution approach, see [Composite Method on page 765](#).

---

### Setting the Excitation Parameters

The Excitation section is common to all optical solvers and mainly specifies a plane wave excitation. It contains the following keywords:

- Intensity [W/cm<sup>2</sup>]
- Wavelength [ $\mu\text{m}$ ]
- Theta [deg]
- Phi [deg]
- PolarizationAngle [deg] or Polarization

## Chapter 21: Optical Generation

### Solving the Optical Problem

In combination with the optical solvers `TMM`, `OptBeam`, and `FromFile`, an additional window section is required to specify the parameters of the illumination window, which is described in [Illumination Window on page 676](#).

#### Note:

For the optical solver `FromFile`, the requirement of a `Window` section only applies when one-dimensional profiles are loaded.

The intensity is defined with respect to an imagined surface normal to the propagation direction of the incident light. For the transfer matrix method (`TMM`) and the optical beam absorption method (`OptBeam`) specifying `ReferenceSurface=IlluminationWindow`, you can set the reference surface to the plane of the illumination window to obtain the default behavior of Version P-2019.03 and earlier.

The propagation direction is defined by the angles with the positive z-axis and x-axis, respectively. In two dimensions, the propagation direction is well-defined by specifying `Theta` only, where `Theta` corresponds to the angle between the propagation direction and the positive y-axis. However, in three dimensions, the propagation direction is defined by `Theta` and `Phi`:

- `Theta` is the angle between the positive z-axis and the propagation direction.
- `Phi` is the angle between the positive x-axis and the projection of the propagation direction on the xy plane as shown in [Figure 30 on page 676](#).

For vectorial methods, such as the FDTD solver, the polarization is set using the keyword `PolarizationAngle`, which represents the angle between the H-field and the  $\hat{z} \times \hat{k}$  axis, where  $\hat{k}$  is the unit wavevector (see *Sentaurus™ Device Electromagnetic Wave Solver User Guide*, Plane Waves).

For the TMM and RT solvers, the conventions for polarization are as follows:

- In two dimensions, the keyword `Polarization` is a real number in the interval [0,1], where `Polarization=0` refers to TM, and `Polarization=1` refers to TE excitations, respectively. If the keyword `PolarizationAngle` is specified, it takes precedence over the keyword `Polarization`. A simple relationship between `PolarizationAngle` and `Polarization` can be established:  $\text{Polarization} = \sin^2(\text{PolarizationAngle})$ .
- In three dimensions, the polarization vector is defined by rotating the  $\hat{z} \times \hat{k}$  vector counterclockwise, about the wave direction, by an angle defined by `PolarizationAngle` when looking in the propagation direction.

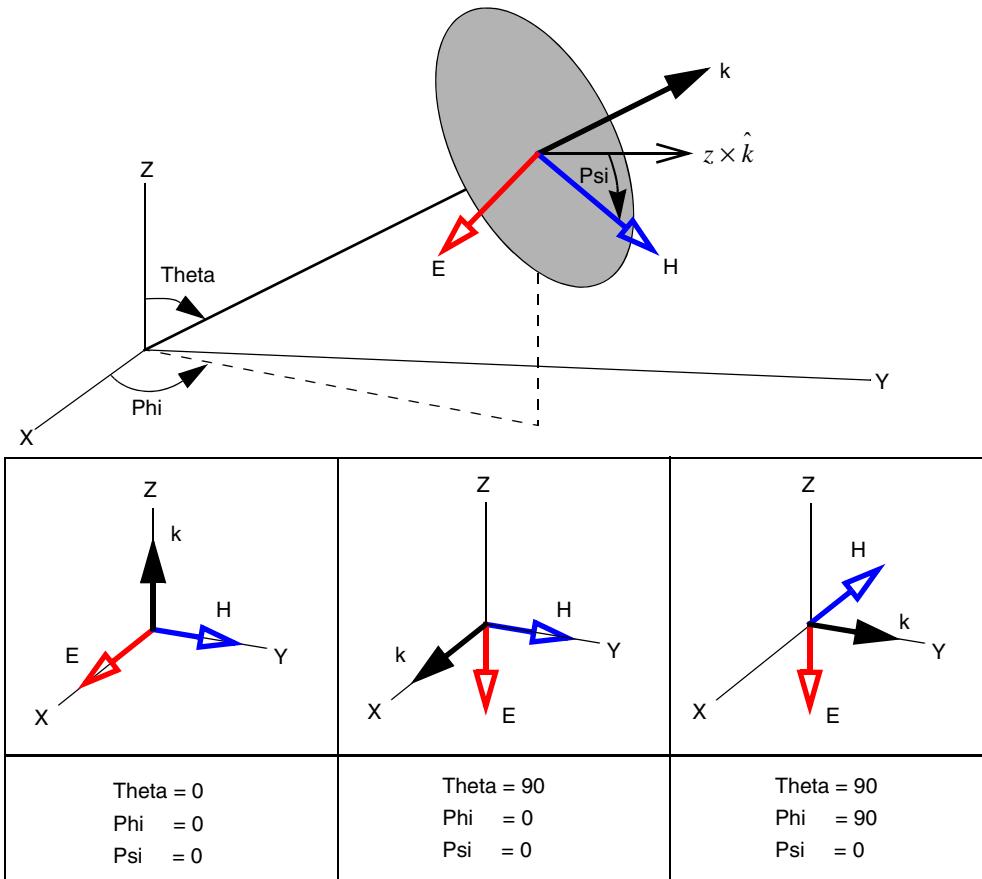
#### Note:

For the RT solver, specifying the `PolarizationVector` explicitly overrides the keywords `Polarization` and `PolarizationAngle`.

## Chapter 21: Optical Generation

### Solving the Optical Problem

**Figure 30** Definition of coordinate system for 3D plane wave excitation and examples of parameters of 3D plane wave



## Illumination Window

The concept of an illumination window to confine the light that is incident on the surface of a device structure plays an important role in various simulation setups for photo-diode devices, such as photodetectors, solar cells, and image sensors. Sentaurus Device supports a flexible user interface for the specification of one or more illumination windows, which also allows you to move these windows during a simulation by ramping the corresponding parameters. This common interface is available for the `TMM`, `FromFile` (only for loading 1D profiles), `OptBeam`, and `RayTracing` (see [Using the Raytracer on page 705](#)) solvers, while different solver-specific implementations exist for `BPM` (see [Using Beam Propagation Method on page 758](#)) solvers.

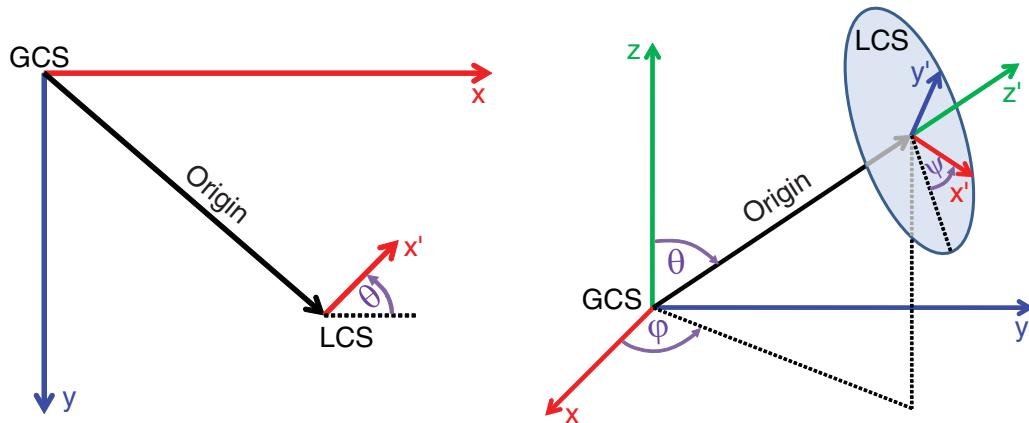
The illumination window is described using a local coordinate system specified by the global location of its origin and the x- and y-directions given as vectors or in terms of rotation angles as illustrated in [Figure 31](#).

## Chapter 21: Optical Generation

### Solving the Optical Problem

Within this local coordinate system, several window shapes can be defined using relative quantities such as width and height, together with an origin anchor for a line in 2D or a rectangular window in 3D. Alternatively, or for shapes that do not support relative quantities, absolute coordinates are used to characterize the shape of the window. For example, a polygon would be defined by specifying a series of vertices in the local 2D coordinate system.

Figure 31 Coordinate system used in the illumination window for (left) two dimensions and (right) three dimensions



The following window shapes are supported:

- Line (in 2D only)
- Rectangle (in 3D only)
- Polygon (in 3D only)
- Circle (in 3D only)

You can define more than one illumination window. The specification of an optional name tag allows you to refer to the respective window when ramping one of its parameters. It is also helpful for identifying results for a specific window in the current file. If two windows overlap, the solutions of the corresponding windows will be added in the intersection.

The global location of the origin of the local coordinate system is specified with the keyword `Origin`, whose default is `(0, 0, 0)`. To specify its orientation, the directions of the coordinate axes can be defined using the vectors `xDirection` and `yDirection`. In two dimensions, only `xDirection` needs to be specified.

Alternatively, you can set three rotation angles `theta` (angle with z-axis), `phi` (angle between x-axis and projection of window normal  $n$  on xy plane), and `psi` (angle between local x-axis and the vector  $z \wedge n$ ), which define the vector `RotationAngles` in three dimensions as shown in [Figure 31](#). For two dimensions, only the first angle of the vector `RotationAngles` is used, which means a rotation of the angle from the local +y'-axis to the

## Chapter 21: Optical Generation

### Solving the Optical Problem

local +x'-axis. The local origin can be defined within each Window using the keyword `Origin=<vector>`. The `RotationAngles` are compatible with the optical solvers TMM, Raytracing, Optbeam, and FromFile. In general, the direction of excitation defined in (`Theta`, `Phi`) of the Excitation section is independent of the angles defined by `RotationAngles`.

If the location of a window in the local coordinate system is not specified by its corresponding vertex coordinates, its placement is determined by the bounding box of the window shape and a cardinal direction such as North, South, NorthWest, and so on, or Center defining which point of the bounding box coincides with the local origin. The cardinal direction is specified using the keyword `OriginAnchor`.

The illumination window is specified in the `Excitation` section as follows:

```
Physics {
    Optics (
        Excitation (
            Window ( <Window options> )
        )
    )
}
```

The following examples introduce the supported window shapes and show different ways of specifying them. Depending on the simulation setup, one specification might be more convenient than others, for example, if the window needs to be moved by using the parameter ramping feature.

#### Line

Line normal to y-axis at y=-10, xmin=-5, xmax=5:

```
Window (
    Origin = (0,-10)
    OriginAnchor = Center # Default
    Line ( Dx = 10 )
)
```

Line normal to y-axis at y=-10, xmin=5, xmax=500:

```
Window (
    Origin = (5,-10)
    OriginAnchor = West
    Line ( Dx = 545 )
)
```

or alternatively:

```
Window (
    Origin = (0,-10)
    XDirection = (1, 0, 0) # Default
    Line (
        X1 = 5
    )
)
```

## Chapter 21: Optical Generation

### Solving the Optical Problem

```
        x2 = 500
    )
)
```

Two lines normal to y-axis at y=-10, with xmin1=5, xmax1=10, and xmin2=15, xmax2=20:

```
Window ("W1") (
    Origin = (5, -10)
    OriginAnchor = West
    Line ( Dx = 5 )
)
Window ("W2") (
    Origin = (15, -10)
    OriginAnchor = West
    Line ( Dx = 5 )
)
```

## Rectangle

Rectangle normal to z-axis at z=10, with width=10 and height=5, centered at x=0 and y=0:

```
Window (
    Origin = (0, 0, 10)
    OriginAnchor = Center # Default
    Rectangle (
        Dx = 10
        Dy = 5
    )
)
```

or alternatively:

```
Window (
    Origin = (0, 0, 10)
    Rectangle (
        Corner1 = (-5, -2.5)
        Corner2 = (5, 2.5)
    )
)
```

## Polygon

Triangle in xz plane at y=4, and corners at (0, 0, 0), (1, 0, 0), and (0, 0, 1):

```
Window (
    Origin = (0, 4, 0)
    XDirection = (1, 0, 0) # Default
    YDirection = (0, 0, 1)
    Polygon( (0, 0), (1, 0), (0, 1) )
)
```

## Chapter 21: Optical Generation

### Solving the Optical Problem

#### Note:

More complex polygon specifications also are supported by specifying several polygon loops, which might or might not intersect. In this case, the vertices of each loop must be enclosed in extra parentheses within the `Polygon` section.

#### Circle

Circle in xy plane with center at (5, 4, -10) and radius 3:

```
Window (
    Origin = (5, 4, -10)
    XDirection = (1, 0, 0) # Default
    YDirection = (0, 1, 0) # Default
    Circle ( Radius = 3 )
)
```

#### Common Illumination Configurations

The most common cases of illumination are from either the top or the bottom of the device structure. Depending on the coordinate system, the DF–ISE coordinate system or the unified coordinate system (UCS), the bottom-to-top growth orientation aligns with the +z-axis or the –x-axis, respectively.

You can also use the `CoordinateSystem` keyword in the `Math` section to choose a different coordinate system from that of the input TDR grid file (see [Reading a Structure on page 57](#)). [Table 121](#) lists the combinations of coordinate systems that you can control.

*Table 121 Combinations of coordinate systems in TDR grid file and Math section*

Coordinate system in input TDR grid file	Math section <code>CoordinateSystem{}</code>	Coordinate system at runtime	Coordinate system in output TDR file
DF–ISE	AsIs	DF–ISE	DF–ISE
DF–ISE	DFISE	DF–ISE	DF–ISE
DF–ISE	UCS	UCS	UCS
UCS	AsIs	UCS	UCS
UCS	UCS	UCS	UCS
UCS	DFISE	DF–ISE	DF–ISE

## Chapter 21: Optical Generation

### Solving the Optical Problem

#### Note:

Excitation and window parameters follow the default coordinate system in Sentaurus Device (that is, the DF–ISE coordinate system). Therefore, if you select the UCS, you must set the correct parameters carefully.

The orientation of the excitation for the UCS might be confusing. Therefore, you can specify a simplified default set of parameters automatically using one of the following options in the Excitation section:

```
Optics (...  
    Excitation (  
        fromTop      # or fromBottom  
    )  
)
```

[Table 122](#) and [Table 123](#) list the parameters affected by specifying `fromTop` or `fromBottom`.

*Table 122 Default values for essential 2D excitation parameters when fromTop or fromBottom is activated*

Excitation parameter	2D DF–ISE coordinate system		2D UCS	
	fromTop	fromBottom	fromTop	fromBottom
Theta	0	180	90	270
Origin	(0,ymin)	(0,ymax)	(xmin,0)	(xmax,0)
XDirection	(1,0)	(1,0)	(0,1)	(0,1)
Window type	Line	Line	Line	Line
x1	xmin	xmin	ymin	ymin
x2	xmax	xmax	ymax	ymax

*Table 123 Default values for essential 3D excitation parameters when fromTop or fromBottom is activated*

Excitation parameter	3D DF–ISE coordinate system		3D UCS	
	fromTop	fromBottom	fromTop	fromBottom
Theta	180	0	90	270
Phi	0	0	0	0

## Chapter 21: Optical Generation

### Solving the Optical Problem

**Table 123** Default values for essential 3D excitation parameters when fromTop or fromBottom is activated (Continued)

Excitation parameter	3D DF-ISE coordinate system		3D UCS	
	fromTop	fromBottom	fromTop	fromBottom
Origin	(0,0,zmax)	(0,0,zmin)	(xmin,0,0)	(xmax,0,0)
XDirection	(1,0,0)	(1,0,0)	(0,0,1)	(0,0,1)
YDirection	(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)
Window type	Rectangle	Rectangle	Rectangle	Rectangle
Corner1	(xmin,ymin)	(xmin,ymin)	(zmin,ymin)	(zmin,ymin)
Corner2	(xmax,ymax)	(xmax,ymax)	(zmax,ymax)	(zmax,ymax)

In these tables, xmin, xmax, ymin, ymax, zmin, and zmax refer to the bounding limits of the device. In two dimensions, Theta denotes an angle that is rotated from the positive x-axis towards the positive y-axis. In three dimensions, Theta starts from the positive z-axis towards the xy plane.

Referring to [Table 122](#) and [Table 123](#), the following examples demonstrate how the syntax is set up automatically.

#### Note:

In these examples, keywords whose default values are correct for the given use case have been omitted to demonstrate the minimum required specification.

DF-ISE coordinate system in two dimensions:

```
Excitation (
    ...
    Window (
        Origin = (0, <ymin of device>)
        Line ( x1 = <xmin> x2 = <xmax> ) # Extent of illumination window
    )
)
```

UCS in two dimensions:

```
Excitation (
    ...
    Theta = 90                  # Illumination from top in +x direction
    Window (
        Origin= (<xmin of device>, 0)
        xDirection= (0, 1)      # Local x' points in global +y direction
    )
)
```

## Chapter 21: Optical Generation

### Solving the Optical Problem

```
        Line (x1=<ymin> x2=<ymax> )    # Extent of illumination window
    )
)
```

DF-ISE coordinate system in three dimensions:

```
Excitation (
    ...
    Theta = 180          # Illumination from top in -z-direction
    Window (
        Origin = (0, 0, <zmax of device>)
        Rectangle (
            # Extent of illumination window
            corner1 = (<xmin>, <ymin>) corner2 = (<xmax>, <ymax> )
        )
    )
)
```

UCS in three dimensions:

```
Excitation (
    ...
    Theta = 90      # Illumination from top in +x-direction
    Window (
        Origin = (<xmin of device>, 0, 0)
        xDirection = (0, 0, 1)  # Local x' points in global +z-direction
        yDirection = (0, 1, 0)  # Local y' points in global +y-direction
        Rectangle (
            # Extent of illumination window
            corner1 = (<zmin>, <ymin>) corner2 = (<zmax>, <ymax> )
        )
    )
)
```

## Moving Illumination Windows Using Parameter Ramping

The following syntax demonstrates how to move and resize illumination windows during a simulation using the parameter ramping framework:

```
Physics {
    Optics (
        Excitation (
            Window("L1") (
                Origin = (-0.5, -0.1, 0)
                XDIRECTION = (1,0,0)           # Default
                Line(
                    x1 = -0.5
                    x2 = 0.5
                )
            )
            Window("L2") (
                Origin = (0.5, -0.1, 0)
                OriginAnchor = Center       # Default
            )
        )
    )
}
```

## Chapter 21: Optical Generation

### Solving the Optical Problem

```

        XDIRECTION = (1,0,0)           # Default
        Line( Dx = 1 )
    )
)
)
}
Solve{
    Optics
    Quasistationary (
        InitialStep=0.2 MaxStep=0.2 MinStep=0.2
        Goal { ModelParameter="Optics/Excitation/Window(L2)/Line/Dx"
            value=0.2 }
        ) { Optics }

    Quasistationary (
        InitialStep=0.2 MaxStep=0.2 MinStep=0.2
        Goal { ModelParameter="Optics/Excitation/Window(L2)/Origin[0]"
            value=0.9 }
        Goal { ModelParameter="Optics/Excitation/Window(L2)/Origin[1]"
            value=0.5 }
        Goal { ModelParameter="Optics/OpticalSolver/TMM/
            LayerStackExtraction(L2)/Position[0]" value=0.9 }
        ) { Optics }
    )
}

```

For details about ramping parameters, including the ramping of vector parameters, see [Parameter Ramping on page 689](#).

## Spatial Intensity Function Excitation

With the illumination window feature, you can define a spatial profile within each illumination window (see [Illumination Window on page 676](#)). The intensity profile corresponds to a modified Gaussian profile with the shape shown in [Figure 32](#).

There is a top plateau, with both ends decaying with symmetric Gaussian functions. Its peak value is normalized to 1.0. Mathematically, the spatial shape is defined by:

$$g_x(x) = \begin{cases} \frac{1}{\sigma\sqrt{\pi}} \exp\left(-\frac{|x - x_{\text{pos}}|^2}{2\sigma^2}\right) & , |x - x_{\text{pos}}| > \frac{x_{\text{width}}}{2} \\ 1 & , \text{otherwise} \end{cases} \quad (697)$$

where  $\sigma$  (`Sigma`) is the Gaussian width,  $x_{\text{pos}}$  (`PeakPosition`) is the center peak position of the function, and  $x_{\text{width}}$  (`PeakWidth`) is the width of the center plateau.

To add flexibility, you can add  $g_0$  (`Scaling`) as a scaling factor to the final expression:

$$g(x) = g_0 \cdot g_x(x) \quad (698)$$

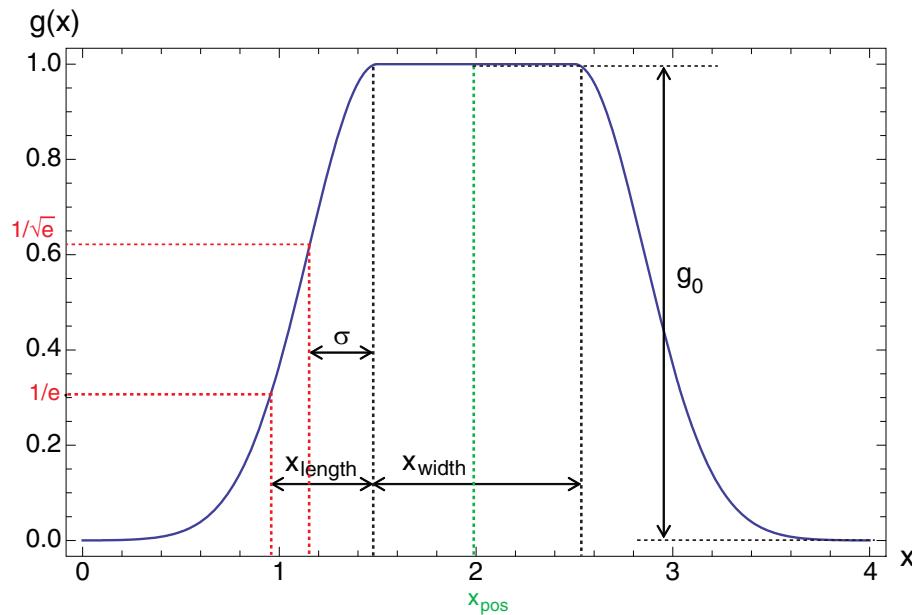
## Chapter 21: Optical Generation

### Solving the Optical Problem

This modified spatial Gaussian function can be applied to the different supported types of illumination window: line, rectangle, polygon, and circle (see [Illumination Window on page 676](#)). In two dimensions, the product of two modified Gaussian functions is:

$$g(x, y) = g_0 \cdot g_x(x) \cdot g_y(y) \quad (699)$$

*Figure 32 Gaussian spatial intensity distribution, showing the definition of key parameters*



The syntax of the spatial intensity function is embedded within the syntax of the illumination window:

```
Optics (
    Excitation (... 
        Wavelength = 0.5
        Intensity = 100      *[W/cm2]
        Window(
            IntensityDistribution(
                Gaussian(
                    Sigma = (<float>,<float>) | Length = (<float>,<float>)
                    PeakPosition = (<float>,<float>)
                    PeakWidth = (<float>,<float>)
                    Scaling = <float>
                )
            )
            Rectangle(dx = 1, dy = 2)
        )
    )
)
```

## Chapter 21: Optical Generation

Solving the Optical Problem

Comments about the syntax:

- The `IntensityDistribution` section is specified within the illumination window (`window`). It follows the local coordinate system of the window, that is, `PeakPosition` is taken with reference to the origin of the local coordinate of the window.
- Either `Sigma` or `Length` must be specified. If `Length` is specified, `Sigma` is calculated using the formula:  $\sigma = x_{\text{length}} / \sqrt{2}$ .
- Parameters are input as a single floating-point number (for one dimension) or a vector (for two dimensions). For example, `Sigma=0.05` for a 1D Gaussian and `Sigma=(0.03, 0.04)` for a 2D Gaussian.
- The `IntensityDistribution` section works with the `raytracing`, `TMM`, `OptBeam`, and `FromFile` optical solvers.

**Note:**

For 1D solvers such as `TMM` and `OptBeam`, the spatial intensity profile is simply overlaid onto the 1D field distribution; it is not an indication of a more accurate solution.

---

## Choosing Refractive Index Model

The complex refractive index model as described in [Complex Refractive Index Model on page 694](#) is available for all optical solvers. It must be specified in the `Optics` section when using the unified interface for optical generation computation.

---

## Extracting the Layer Stack

The optical solvers `TMM` (see [Transfer Matrix Method on page 669](#)) and `OptBeam` (see [Optical Beam Absorption Method on page 753](#)) require the extraction of a layer stack from the actual device structure as they are based on 1D algorithms whose solution is extended to the dimension of the simulation grid. Both the extension of the solution and the extraction of the layer stack are bound to the illumination window (see [Illumination Window on page 676](#)), which determines the domain of the device structure being represented by the 1D optical problem.

The layer stack extraction algorithm works by extracting all grid elements along a line normal to the corresponding illumination window starting from a user-specified position within the window. The direction of extraction, that is, positive or negative, is derived from the propagation direction of the incident light as specified in the `Excitation` section.

By default, all elements belonging to the same region will form a single layer, which is part of the entire stack. This assumes that the material properties do not vary within the region. If this is not the case or not a good approximation, you can create a separate layer for each

## Chapter 21: Optical Generation

### Solving the Optical Problem

extracted element by selecting the option `ElementWise` instead of `RegionWise` for the keyword `Mode` in the `LayerStackExtraction` section. As this can lead to a large number of layers whose difference in material properties might be insignificant for the solution of the optical solver, yet impact its performance, a threshold value can be specified to control the creation of the layer stack from the extracted grid elements.

All elements whose relative difference in the refractive index or extinction coefficient, compared with the previous layer, is lower than the threshold are merged into one layer. In doing so, the refractive index and the extinction coefficient are treated separately, both deviations must be lower than the specified threshold. The material properties of this merged layer are obtained by averaging the corresponding element properties. The thickness of each element computed by the distance between the intersection points of the extraction line with the corresponding element is summed to yield the total thickness of the representative layer of the stack.

The threshold can be specified using the keyword `ComplexRefractiveIndexThreshold` in the `LayerStackExtraction` section.

Options are available for defining the starting point of the extraction line:

- Exact position in the global coordinate system using the keyword `Position`
- Position within the bounding box of the window designated by a cardinal direction
- Position within the bounding box of the window specified as a point in the local coordinate system of the illumination window

The last two options require the specification of the keyword `WindowPosition`. Its argument can be either one of `North`, `South`, `NorthWest`, and so on, and `Center` or a 2D point, for example, `WindowPosition=(1.5, 2)`.

The following is an example for a `LayerStackExtraction` section used by the TMM solver:

```
Physics {
    Optics (
        OpticalSolver (
            TMM (
                LayerStackExtraction (
                    WindowName = "L1"
                    Position = (-0.5, -0.1, 0)
                    Mode = ElementWise # Default RegionWise
                )
            )
        )
    )
}
```

The keyword `WindowName` is required if more than one illumination window exists. Its argument specifies to which illumination window the `LayerStackExtraction` refers.

## Chapter 21: Optical Generation

Solving the Optical Problem

### Note:

If the starting point of the extraction line is outside of the device structure, the extracted layer stack will contain a layer representing the space between the starting point of the extraction line and the surface of the device structure. In general, the material properties of vacuum are assigned to this layer. For optical solvers that support user-defined bulk media on the top and the bottom of the layer stack using the `Medium` statement, the material properties of the top medium if specified are used except that the extinction coefficient `k` is set to 0. For details about user-defined bulk media, see [Using Transfer Matrix Method on page 739](#).

---

## Controlling Computation of Optical Problem in Solve Section

Specifying the keyword `Optics` in the `Solve` section triggers the solution of the optical problem. For example, the following is sufficient to compute the optical generation and the optical intensity:

```
Physics {  
    Optics (  
        OpticalGeneration ( ... )  
    )  
}  
  
Solve { Optics }
```

If only the keyword `Optics` is specified in the `Solve` section, either directly or within a `Quasistationary` or `Transient` statement, and no other equations are solved, the simulation is classified as an optics standalone simulation. In general, such simulations do not require the specification of contacts in the grid file. In optics standalone simulations, it is possible to switch on the creation of double points at region interfaces independent of the type of interface. In other words, a heterojunction interface is treated in the same way as a semiconductor–insulator interface. This feature is enabled by specifying the keyword `Discontinuity` in the `Physics` section (see [Discontinuous Interfaces on page 288](#)).

The `Solve` section of a typical transient device simulation reads as follows:

```
Solve {  
    Optics  
    Poisson  
    Coupled { Poisson Electron Hole }  
  
    Transient (  
        InitialTime = 0  
        FinalTime = 3e-6  
    ) { Coupled { Poisson Electron Hole } }  
}
```

In this example, the optical generation rate is computed at the beginning when the optical problem is solved and is used afterwards in the electronic equations. To recompute the

## Chapter 21: Optical Generation

### Parameter Ramping

optical generation at a later point, the `Optics` statement can be listed wherever a `Coupled` statement is allowed.

The optical generation depends on the solution of the optical problem and, therefore, has an implicit dependency on all quantities that serve as input to the optical equation. Internally, an automatic update scheme ensures that, whenever the optical generation rate is read, all its underlying variables are up-to-date. Otherwise, it will be recomputed. This mechanism can be switched off by specifying `-AutomaticUpdate` in the `OpticalGeneration` section. By default, `AutomaticUpdate` is switched on.

---

## Parameter Ramping

Sentaurus Device can be used to ramp the values of physical parameters (see [Ramping Physical Parameter Values on page 126](#)). For example, it is possible to sweep the wavelength of the incident light to extract the spectral dependency of a certain output characteristic conveniently. Ramping model parameters of the unified interface for optical generation computation works the same way except that instead of using the expression `Parameter= "<parameter name>"`, it uses the keyword `ModelParameter= "<parameter name or path>"`.

The value of `ModelParameter` can be either the parameter name itself such as `"Wavelength"` if it is unambiguous or the parameter name including its full path:

```
Solve {
    Quasistationary (
        Goal {
            [ Device = <device> ]
            [ Material = <material> | MaterialInterface = <interface> |
              Region = <region> | RegionInterface = <interface> ]
            ModelParameter = "Optics/Excitation/Wavelength" Value =
              <float>
        }
    ){ Optics }
```

Specifying the device and location (material, material interface, region, or region interface where applicable) is optional. However, `ModelParameter` and its value must always be specified.

Similarly, the corresponding `CurrentPlot` section reads as follows:

```
CurrentPlot {
    [ Device = <device> ]
    [ Material = <material> | MaterialInterface = <interface> |
      Region = <region> | RegionInterface = <interface> ]
    ModelParameter = "Optics/Excitation/Wavelength"
}
```

## Chapter 21: Optical Generation

### Parameter Ramping

Parameters whose value type is a vector of floating-point numbers support ramping of the individual vector components.

For example, to ramp the x- and y-components of the illumination window origin, the following syntax is required:

```
Quasistationary (
    InitialStep=0.2 MaxStep=0.2 MinStep=0.2
    Plot { Range = ( 0., 1 ) Intervals = 5 }
    Goal { ModelParameter="Optics/Excitation/Window(L2)/Origin[0]"
           value=0.9 }
    Goal { ModelParameter="Optics/Excitation/Window(L2)/Origin[1]"
           value=0.5 }
) {
    Optics
}
```

#### Note:

When ramping parameters that reside in sections that can occur multiple times, such as the Window section, an identifying section name or tag in parentheses must follow the section name in the parameter path as shown in the example above.

*Table 124 Parameters that can be ramped using the unified interface for optical generation computation*

Parameter path	Parameter name
Optics/OpticalGeneration	Scaling
Optics/OpticalGeneration/ComputeFromMonochromaticSource	Scaling
Optics/OpticalGeneration/ReadFromFile	Scaling
Optics/OpticalGeneration/ComputeFromSpectrum	Scaling
Optics/OpticalGeneration/SetConstant	Value
Optics/Excitation	Wavelength, Intensity, Theta, Phi, PolarizationAngle, Polarization
Optics/Excitation/Window	All parameters that are not of type string or identifier
Optics/OpticalSolver/<SolverName>	Except for raytracing, all parameters that are not of type string or identifier

## Chapter 21: Optical Generation

### Accurate Absorbed Photon Density for 1D Optical Solvers

A list of parameter names that can be ramped in the command file is printed when the following command is executed:

```
sdevice --parameter-names <command file>
```

**Note:**

Parameters used in ramping must also exist in the relevant `Physics` section in order to set the values of the parameters to the initial values specified in the `Physics` section.

---

## Accurate Absorbed Photon Density for 1D Optical Solvers

In 1D optical solvers such as `TMM`, `OptBeam`, and `FromFile`, the 1D absorbed photon density (APD) is superimposed onto the projected volume from an illumination window that extends into the 2D or 3D device geometry.

While the sampled APD at each vertex is exact for the 1D solver, the integration with `CurrentPlot` simply multiplies the box method volume, and this can result in inaccuracy due to the following:

- If the mesh is coarse, the projected volume from an illumination window intersects the box method volumes partially at the fringes.
- Depending on where the vertices are located, the box method–integrated APD can be smaller or larger than the true integrated value.

Such inaccuracies can lead to nonconservation of particles or energy because an incorrectly integrated APD produces an incorrect optically generated current.

To reduce the mesh dependency of the integrated APD values for 1D optical solvers, a virtual wireframe is constructed over the projected volume from an illumination window to compute a discrete Riemann integral of the APD. In the transverse plane of the illumination window, the wireframe is chosen automatically with either a discretization of 10 nm or a limit of 200 transverse wireframe cells. In the propagation direction, the wireframe is first divided into layers and, then within each layer, a further discretization of `NumberOfCellsPerLayer` is used for refinement.

The number of layers for the `TMM` and `OptBeam` solvers is fixed at the number of extraction layers, while the `FromFile` optical solver allows you to specify the number of layers.

`NumberOfCellsPerLayer` can be controlled by all three optical solvers. Although a denser wireframe provides a more accurate integrated APD, it also slows down the simulation since the APD must be sampled at every wireframe point in the propagation direction.

After the accurate integrated APD has been computed, it is used to compute a weighting factor for the APD on the mesh vertices such that, when `CurrentPlot` multiplies the box method volume to obtain the integrated values, you can retrieve the correct integrated APD.

## Chapter 21: Optical Generation

### Accurate Absorbed Photon Density for 1D Optical Solvers

The vertices within the projected volume from an illumination window are divided into the center zone and the fringe zone:

- Center zone vertices have edges that connect only to vertices within the projected window volume.
- Fringe zone vertices have edges that cross the boundary of the projected window volume.

#### Note:

The weighting factor is applied only to the fringe zone vertices, so you might see a distorted APD value at those vertices.

Numerically, the weighting factor is computed in each layer and is defined as:

$$W_{\text{layer}(k)} = \frac{\geq f_{\text{APD}}(\text{wireframe})dv - F_{\text{APD}}(v_i) \cdot \text{box}(v_i)}{\frac{\text{centerzone}}{F_{\text{APD}}(v_i) \cdot \text{box}(v_i)} + \frac{\text{fringezone}}{F_{\text{APD}}(v_i) \cdot \text{box}(v_i)}} \quad (700)$$

After this computation, the weighting factor is applied to the APD of the fringe zone vertices:

$$F'_{\text{APD}}(v_i) = W_{\text{layer}(k)} \cdot F_{\text{APD}}(v_i) \quad (701)$$

However, there can be instances whereby the center zone box-integrated APD is greater than the wireframe-integrated APD. In such cases, the weighting factor is computed as:

$$W_{\text{layer}(k)} = \frac{\geq f_{\text{APD}}(\text{wireframe})dv}{\frac{\text{centerzone}}{F_{\text{APD}}(v_i) \cdot \text{box}(v_i)} + \frac{\text{fringezone}}{F_{\text{APD}}(v_i) \cdot \text{box}(v_i)}} \quad (702)$$

and it is applied to modify the APD of all center and fringe zone vertices.

This new methodology can be activated selectively in different illumination windows as follows:

```
Physics {...  
    Optics (...  
        Excitation (...  
            Window("win_1") (...  
                # To switch off, add '-' in front or leave blank.  
                -WeightedAPDintegration  
            )  
            Window("win_2") (...  
                # To switch on, use this keyword  
                WeightedAPDintegration  
            )  
        )  
    )  
}
```

## Chapter 21: Optical Generation

### Accurate Absorbed Photon Density for 1D Optical Solvers

The syntax for controlling the wireframe density is:

```
Physics {...  
    Optics (...  
        Excitation (...  
            Window(...) (...  
                # Various options available  
                WeightedAPDIntegration(  
                    PrintInfo  
                    NumberOfCellsPerLayer=<int>                      # default: 10000  
                    NumberOfTransverseCellsPerDirection=<int> # default: 200  
                    Mode = Auto                                # or Full, default: Auto  
                )  
            )  
        )  
    )  
    OpticalSolver(...  
        FromFile(...  
            # Default is the number of regions but is limited to 20  
            WeightedAPDIntegrationLayers=<int>  
            WeightedAPDIntegrationSetBoundaries= (<float>, <float>, ...)  
        )  
    )  
}  
}
```

Some comments about this syntax:

- To speed up the wireframe integration of APD, you probe strategic points (three points in two dimensions and five points in three dimensions) in each delta projection window (the illumination window area multiplied by the wireframe cell thickness) to determine whether to perform the full transverse numeric integration or to use a precomputed volume. The keyword `Mode=Auto` allows for a quick assessment and should be used when the projected illumination window is expected to stay within the device domain.
- For the `TMM` and `OptBeam` optical solvers, the number of layers is set to be the same as that from the `LayerStackExtraction` section. For the `FromFile` optical solver, you can set the number of layers using `WeightedAPDIntegrationLayers`, which divides the total propagation length into equal layer thicknesses. Alternatively, if the structure contains important thin layers, you can specify the boundaries of each layer using `WeightedAPDIntegrationSetBoundaries`. These boundaries are measured from the illumination window in the direction of propagation, so they must be positive numbers. Otherwise, the default for the number of layers is the number of significant regions, that is, regions whose volumes are at least 10% of the total volume of the device.
- You can control the discretization of the wireframe with the following keywords:
  - `NumberOfCellsPerLayer` controls the discretization in the direction of propagation.
  - `NumberOfTransverseCellsPerDirection` controls the discretization of the illumination window in the 2D plane containing the window.

**Note:**

The plane of the window might not necessarily align with the standard xy, yz, or xz planes.

---

## Complex Refractive Index Model

The complex refractive index model in Sentaurus Device allows you to define the refractive index and the extinction coefficient depending on mole fraction, wavelength, temperature, carrier density, and local material gain. In addition, it provides a flexible interface that can be used to add new complex refractive index models as a function of almost any internally available variable. See [Complex Refractive Index Model Interface on page 1360](#).

The complex refractive index model is designed primarily for use in the unified interface for optical generation computation and LED simulations. Other tools besides Sentaurus Device, such as Sentaurus Device Electromagnetic Wave Solver (EMW) and Sentaurus Mesh, also support this model, allowing for a consistent source of optical material parameters across the entire tool flow. A common Sentaurus Device parameter file can be shared and only the syntax for activating the different models might slightly change from tool to tool to comply with the respective command file syntax.

---

## Physical Model

The complex refractive index  $\tilde{n}$  can be written as:

$$\tilde{n} = n + i \cdot k \quad (703)$$

with:

$$n = n_0 + \Delta n_\lambda + \Delta n_T + \Delta n_{\text{carr}} + \Delta n_{\text{gain}} \quad (704)$$

$$k = k_0 + \Delta k_\lambda + \Delta k_{\text{carr}} \quad (705)$$

The real part  $n$  is composed of the base refractive index  $n_0$ , and the correction terms  $\Delta n_\lambda$ ,  $\Delta n_T$ ,  $\Delta n_{\text{carr}}$ , and  $\Delta n_{\text{gain}}$ . The correction terms include the dependency on wavelength, temperature, carrier density, and gain.

The imaginary part  $k$  is composed of the base extinction coefficient  $k_0$ , and the correction terms  $\Delta k_\lambda$  and  $\Delta k_{\text{carr}}$ . The correction terms include the dependency on wavelength and carrier density. The absorption coefficient  $\alpha$  is computed from  $k$  and wavelength  $\lambda$  according to:

$$\alpha = \frac{4\pi}{\lambda} \cdot k \quad (706)$$

## Wavelength Dependency

The complex refractive index model offers different ways to take wavelength dependency into account:

- An analytic formula considers a linear and square dependency on the wavelength  $\lambda$ :

$$\begin{aligned}\Delta n_\lambda &= C_{n,\lambda} \cdot \lambda + D_{n,\lambda} \cdot \lambda^2 \\ \Delta k_\lambda &= C_{k,\lambda} \cdot \lambda + D_{k,\lambda} \cdot \lambda^2\end{aligned}\quad (707)$$

You can adjust the  $C_{n,\lambda}$ ,  $D_{n,\lambda}$ ,  $C_{k,\lambda}$ , and  $D_{k,\lambda}$  parameters in the ComplexRefractiveIndex section of the parameter file (see [Table 127 on page 698](#)).

- Tabulated values can be read from the parameter file. The table contains three columns specifying wavelength  $\lambda$ , refractive index  $n'$ , and the extinction coefficient  $k'$ . Thereby,  $n'$  and  $k'$  represent  $n_0 + \Delta n_\lambda$  and  $k_0 + \Delta k_\lambda$ , respectively. The character \* is used to insert a comment. The row is terminated by a semicolon. For wavelengths not listed in the table, the refractive index and the extinction coefficient are obtained using linear interpolation or spline interpolation. At least two rows are required for linear interpolation. To form a cubic spline from the table entries, at least four rows are needed.
- Tabulated values can be read from an external file. The file holds a table that is structured as described in the previous point. The name of the file is specified in the ComplexRefractiveIndex section of the parameter file.

## Temperature Dependency

The temperature dependency of the real part of the complex refractive index follows the relation according to:

$$\Delta n_T = n_0 \cdot C_{n,T} \cdot (T - T_{\text{par}}) \quad (708)$$

The parameters  $C_{n,T}$  and  $T_{\text{par}}$  can be adjusted in the ComplexRefractiveIndex section of the parameter file (see [Table 130 on page 699](#)).

## Carrier Dependency

The change in the real part of the complex refractive index due to free carrier absorption is modeled according to [1]:

$$\Delta n_{\text{carr}} = -C_{n,\text{carr}} \cdot \frac{q^2 \lambda^2}{8\pi^2 c^2 \epsilon_0 n_0} \cdot \frac{n}{m_n} + \frac{p}{m_p} \quad (709)$$

where:

- $C_{n,\text{carr}}$  is a fitting parameter.
- $q$  is the elementary charge.

## Chapter 21: Optical Generation

### Complex Refractive Index Model

- $\lambda$  is the wavelength.
- $c$  is the speed of light in free space.
- $\epsilon_0$  is the permittivity.

Furthermore,  $n$  and  $p$  are the electron and hole densities, and  $m_n$  and  $m_p$  are the effective masses of electron and hole, which are computed according to the specification of the eDOSMass and hDOSMass sections in the parameter file. If only optics is solved, for example `Solve {Optics}` is specified in the command file, then  $n$  and  $p$  correspond to the respective doping concentrations.

To account for the change of the extinction coefficient due to free carrier absorption, a model is available that assumes linear dependency on carrier concentration and power-law dependency on wavelength:

$$\Delta k_{\text{carr}} = \frac{10^{-4}}{4\pi} \cdot \frac{\lambda}{\mu\text{m}} \left( \frac{\Gamma_{k,\text{carr},n} C_{k,\text{carr},n}}{\text{cm}^2} + \frac{\Gamma_{k,\text{carr},p} C_{k,\text{carr},p}}{\text{cm}^2} \right) \quad (710)$$

where:

- $C_{k,\text{carr},n}$ ,  $C_{k,\text{carr},p}$ ,  $\Gamma_{k,\text{carr},n}$ ,  $\Gamma_{k,\text{carr},p}$  are fitting parameters.
- $\lambda$  is the wavelength in  $\mu\text{m}$ .
- $n$  and  $p$  are the electron and hole densities in units of  $\text{cm}^{-3}$ .

For linear dependency on wavelength,  $C_{k,\text{carr},n}$  and  $C_{k,\text{carr},p}$  are given in units of  $\text{cm}^{-2}$ .

An alternative formulation found in the literature [2][3] reads as follows:

$$\Delta\alpha_{\text{FCA}} = An\lambda^B + Cp\lambda^D \quad (711)$$

where the free carrier absorption coefficient  $\Delta\alpha_{\text{FCA}}$  is given in units of  $\text{cm}^{-1}$ , the wavelength is given in nanometers, and the carrier concentrations are given in  $\text{cm}^{-3}$ .

The fitting parameters  $A$ ,  $B$ ,  $C$ , and  $D$  in [Equation 711](#) are related to the corresponding fitting parameters in [Equation 710](#) by the following expressions:

$$\Gamma_{k,\text{carr},n} = B + 1 \quad (712)$$

$$C_{k,\text{carr},n} = 10^{3B} A \quad (713)$$

$$\Gamma_{k,\text{carr},p} = D + 1 \quad (714)$$

$$C_{k,\text{carr},p} = 10^{3D} C \quad (715)$$

You can adjust the parameters  $C_{n,\text{carr}}$ ,  $C_{k,\text{carr},n}$ ,  $C_{k,\text{carr},p}$ ,  $\Gamma_{k,\text{carr},n}$ , and  $\Gamma_{k,\text{carr},p}$  in the ComplexRefractiveIndex section of the parameter file (see [Table 131 on page 699](#)).

## Gain Dependency

The complex refractive index model offers different formulas to take into account the gain dependency:

- The linear model is given by  $\Delta n_{\text{gain}} = C_{n, \text{gain}} \cdot \frac{n + p}{2N_{\text{par}}} - 1$ .
- The logarithmic model reads  $\Delta n_{\text{gain}} = C_{n, \text{gain}} \cdot \ln \frac{n + p}{2N_{\text{par}}}$ .

You can adjust the parameters  $C_{n, \text{gain}}$  and  $N_{\text{par}}$  in the `ComplexRefractiveIndex` section of the parameter file (see [Table 132 on page 700](#)).

You can implement additional complex refractive models with arbitrary dependencies by using the complex refractive index model interface (see [Complex Refractive Index Model Interface on page 1360](#)).

---

## Using Complex Refractive Index

The complex refractive index model is activated by using the `ComplexRefractiveIndex` statement in the `Optics` section of the `Physics` section. When using the unified interface for optical generation computation, `ComplexRefractiveIndex` can be specified globally, per region or per material. Otherwise, support is only available for global specification.

[Table 288 on page 1671](#) lists the available keywords, which allow you to select the dependencies that change the complex refractive index. For example:

```
Physics {
    Optics (
        ComplexRefractiveIndex (
            WavelengthDep (real imag)
            TemperatureDep (real)
            CarrierDep (real imag)
            GainDep (real(log))
            CRIModel(Name=<string>)
        )
    )
}
```

[Table 125](#) to [Table 132](#) summarize all the parameters valid for the `ComplexRefractiveIndex` section of the parameter file. They can be specified for mole-dependent materials using the standard Sentaurus Device technique with linear interpolation on specified mole intervals.

Interpolation for mole-dependent tables is more complex due to the 2D nature of the problem (interpolation with respect to mole fraction and wavelength) and the sharp discontinuities near the absorption edges. Therefore, if `Formula` = 1–3, the complex refractive index at a given wavelength and mole fraction is computed by interpolating its

## Chapter 21: Optical Generation

### Complex Refractive Index Model

value with respect to the wavelength for the lower and upper limits of the mole-fraction interval and subsequently with respect to the mole fraction. Optionally, the latter interpolation can be linear or piecewise constant. By default, mole-fraction interpolation for `NumericalTable` is switched off, and Sentaurus Device will exit with an error if the complex refractive index is requested at a mole fraction for which no `NumericalTable` is defined.

Table 125 Base parameters

Symbol	Parameter name	Unit
$n_0$	<code>n_0</code>	1
$k_0$	<code>k_0</code>	1

The keyword `Formula` is used in the parameter file to select one of the available models to describe the wavelength dependency.

Table 126 Model selection for wavelength dependency

Keyword	Value	Description
<code>Formula</code>	=0	Use analytic formula (default).
	=1	Read tabulated values from parameter file.
	=2	Read tabulated values from external file.

Table 127 Parameters used to described wavelength dependency

Symbol	Parameter name	Unit
$C_{n,\lambda}$	<code>Cn_lambda</code>	$\mu\text{m}^{-1}$
$D_{n,\lambda}$	<code>Dn_lambda</code>	$\mu\text{m}^{-2}$
$C_{k,\lambda}$	<code>Ck_lambda</code>	$\mu\text{m}^{-1}$
$D_{k,\lambda}$	<code>Dk_lambda</code>	$\mu\text{m}^{-2}$

Table 128 lists the different interpolation methods available for `NumericalTable`, where *logarithmic interpolation* refers to taking the natural logarithm of the coordinates, performing linear interpolation, and exponentiating the results. If a value must be interpolated in an interval where one or both of the coordinates at the interval boundaries is smaller than or equal to zero, linear interpolation is used instead.

## Chapter 21: Optical Generation

### Complex Refractive Index Model

*Table 128 Specifying interpolation method for NumericalTable*

Keyword	Value	Description
TableInterpolation	=Linear	Use linear interpolation.
	=Logarithmic	Use logarithmic interpolation.
	=PositiveSpline	Use cubic spline interpolation. If interpolated values are negative, set them to zero (default).
	=Spline	Use cubic spline interpolation.

[Table 129](#) lists the different interpolation methods available for interpolation of mole fraction-dependent NumericalTable.

*Table 129 Specifying interpolation method for mole fraction-dependent NumericalTable*

Keyword	Value	Description
MolefractionTableInterpolation	=Linear	
	=Off	Default
	=PiecewiseConstant	

*Table 130 Parameters used to describe temperature dependency*

Symbol	Parameter name	Unit
$C_{n, T}$	Cn_temp	$K^{-1}$
$T_{\text{par}}$	Tpar	K

*Table 131 Parameters used to describe carrier dependency*

Symbol	Parameter name	Unit	Description
$C_{n, \text{carr}}$	Cn_carr	1	
$C_{k, \text{carr}, n}, C_{k, \text{carr}, p}$	Ck_carr	$\text{cm}^2$	A comma separates values for electrons and holes.

## Chapter 21: Optical Generation

### Complex Refractive Index Model

*Table 131 Parameters used to describe carrier dependency (Continued)*

Symbol	Parameter name	Unit	Description
$\Gamma_{k, \text{carr}, n}, \Gamma_{k, \text{carr}, p}$	Gamma_k_carr	1	A comma separates values for electrons and holes.

*Table 132 Parameters used to described gain dependency*

Symbol	Parameter name	Unit
$C_{n, \text{gain}}$	Cn_gain	1
$N_{\text{par}}$	Npar	$\text{cm}^{-3}$

An example of the `ComplexRefractiveIndex` section in the parameter file is:

```
ComplexRefractiveIndex {
    * Complex refractive index model: n_complex = n + i*k (unitless)
    * Base refractive index and extinction coefficient
    n_0 = 3.45      # [1]
    k_0 = 0.00      # [1]

    * Wavelength dependence
    * Example for analytical formula:
    Formula = 0
    Cn_lambda = 0.0000e+00      # [um^-1]
    Dn_lambda = 0.0000e+00      # [um^-2]
    Ck_lambda = 0.0000e+00      # [um^-1]
    Dk_lambda = 0.0000e+00      # [um^-2]
    * Example for reading values from parameter file:
    Formula = 1
    TableInterpolation = Spline
    NumericalTable
        ( * wavelength [um]      n [1]      k [1]
          0.30      5.003      4.130;
          0.31      5.010      3.552;
          0.32      5.023      3.259;
          0.33      5.053      3.009;
        )
    * Example for reading values from external file:
    Formula = 2
    NumericalTable = "GaAs.txt"

    * Temperature dependence (real):
    Cn_temp = 2.0000e-04      # [K^-1]
    Tpar = 3.0000e+02         # [K]

    * Carrier dependence (real)
```

## Chapter 21: Optical Generation

### Complex Refractive Index Model

```
Cn_carr = 1                                # [1]
* Carrier dependence (imag)
Ck_carr = 0.0000e+00 , 0.0000e+00      # [cm^2]

* Gain dependence (real)
Cn_gain = 0.0000e+00      # [cm^3]
Npar = 1.0000e+18        # [cm^-3]
}
```

The following `ComplexRefractiveIndex` section demonstrates the use of mole fraction-dependent NumericalTables:

```
ComplexRefractiveIndex {
    Formula = 1
    TableInterpolation = Linear
    MolefractionTableInterpolation = Linear
    Xmax(0)=0.0
    NumericalTable(0)
    ( * wavelength [um]    n [1]    k [1]
      0.30      5.003    4.130;
      0.31      5.010    3.552;
      0.32      5.023    3.259;
      0.33      5.053    3.009;
    )
    Xmax(1)=0.25
    NumericalTable(1)
    ( * wavelength [um]    n [1]    k [1]
      0.20      5.003    4.130;
      0.29      4.010    2.552;
      0.34      4.023    1.259;
      0.49      4.053    2.009;
    )
    Xmax(2)=0.6
    NumericalTable(2)
    ( * wavelength [um]    n [1]    k [1]
      0.25      3.123    0.130;
      0.27      3.410    0.552;
      0.29      3.523    0.259;
      0.43      3.753    0.009;
    )
}
```

The complex refractive index is plotted when the keyword `ComplexRefractiveIndex` is defined in the `Plot` section.

#### Note:

`TableODB` is no longer supported. A wavelength table of complex refractive indices can be input using the `ComplexRefractiveIndex -NumericalTable` section instead.

## Chapter 21: Optical Generation

### Raytracing

The complex refractive index model is available for the following optical solvers:

- Raytracing (see [Raytracing](#))
- Transfer matrix method for optical generation computation (see [Transfer Matrix Method on page 669](#))

---

## Raytracing

Sentaurus Device supports the simulation of photogeneration by raytracing in two and three dimensions for arbitrarily shaped structures, as well as body-of-revolution structures using 2D and cylindrical coordinates. The calculation of refraction, transmission, and reflection follows geometric optics, and special boundary conditions can be defined. A dual-grid setup can be used to speed up simulations that include raytracing.

---

### Raytracer

In Sentaurus Device, the raytracer has been implemented based on linear polarization. It is optimized for speed and needs to be used in conjunction with the complex refractive index model (see [Complex Refractive Index Model on page 694](#)). Each region/material must have a complex refractive index section defined in the parameter file. If the refractive index is zero, it is set to a default value of 1.0.

The raytracer uses a recursive algorithm: It starts with a source ray and builds a binary tree that tracks the transmission and reflection of the ray. A reflection/transmission process occurs at interfaces with refractive index differences. This is best illustrated in [Figure 33](#).

An incident ray impinges on the interface of two different refractive index ( $n_1$  and  $n_2$ ) regions, resulting in a reflected ray and a transmitted ray. The incident, reflected, and transmitted rays are denoted by the subscripts  $i$ ,  $r$ , and  $t$ , respectively. Likewise, the incident, reflected, and transmitted angles are denoted by  $\theta_i$ ,  $\theta_r$ , and  $\theta_t$ , respectively. These angles can be derived from the concept of interface tangential phase-matching (commonly called Snell's law) using:

$$n_1 \sin \theta_i = n_2 \sin \theta_t \quad (716)$$

To define these angles, a plane of incidence must be clearly defined. It is apparent that the plane of incidence is the plane that contains both the normal to the interface and the vector of the ray.

When the plane of incidence is defined, the concept of TE and TM polarization can then be established.

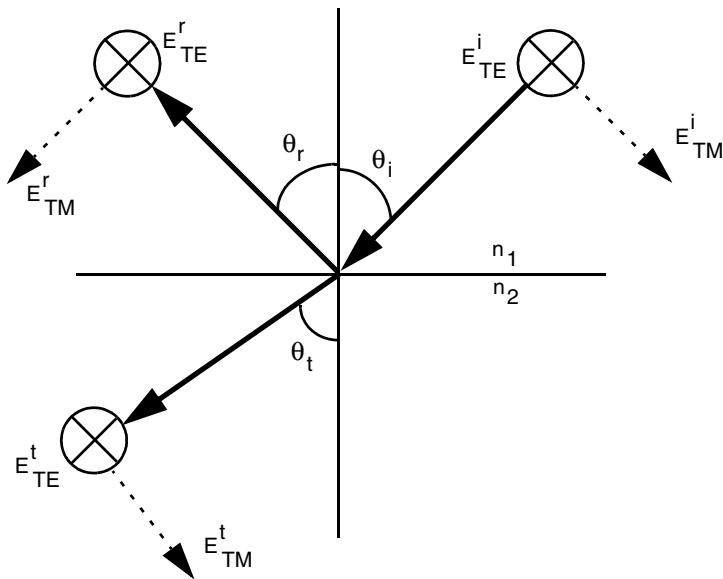
A ray can be considered a plane wave traveling in a particular direction with its polarization vector perpendicular to the direction of propagation. The length of the polarization vector

## Chapter 21: Optical Generation

### Raytracing

represents the amplitude, and the square of its length denotes the intensity. The TE polarization (s-wave) applies to the ray polarization vector component that is perpendicular to the plane of incidence. On the other hand, the TM polarization (p-wave) applies to the ray polarization vector component that is parallel to the plane of incidence.

**Figure 33** Incident ray splits into reflected and transmitted rays at an interface: the TE component of the polarization vector maintains the same direction, whereas the TM component changes direction



In Figure 33, the TE and TM components of the ray polarization vector are denoted by  $E_{TE}$  and  $E_{TM}$ , respectively.

The TE and TM components of the ray polarization vector experience different reflection and transmission coefficients. These coefficients are:

Amplitude reflection coefficients:

$$r_{TE} = \frac{k_{1z} - k_{2z}}{k_{1z} + k_{2z}} \quad (717)$$

$$r_{TM} = \frac{\epsilon_2 k_{1z} - \epsilon_1 k_{2z}}{\epsilon_2 k_{1z} + \epsilon_1 k_{2z}} \quad (718)$$

Amplitude transmission coefficients:

$$t_{TE} = \frac{2k_{1z}}{k_{1z} + k_{2z}} \quad (719)$$

$$t_{TM} = \frac{2\epsilon_2 k_{1z}}{\epsilon_2 k_{1z} + \epsilon_1 k_{2z}} \quad (720)$$

## Chapter 21: Optical Generation

### Raytracing

Power reflection coefficients:

$$R_{TE} = |r_{TE}|^2 \quad (721)$$

$$R_{TM} = |r_{TM}|^2 \quad (722)$$

Power transmission coefficients:

$$T_{TE} = \frac{k_{2z}}{k_{1z}} |t_{TE}|^2 \quad (723)$$

$$T_{TM} = \frac{\epsilon_1 k_{2z}}{\epsilon_2 k_{1z}} |t_{TM}|^2 \quad (724)$$

where:

$$k_0 = 2\pi/\lambda_0 \quad (725)$$

$$k_{1z} = n_1 k_0 \cos \theta_i \quad (726)$$

$$k_{2z} = n_2 k_0 \cos \theta_t \quad (727)$$

$$\epsilon_1 = n_1^2 \quad (728)$$

$$\epsilon_2 = n_2^2 \quad (729)$$

where  $\lambda_0$  is the free space wavelength, and  $k_0$  is the free space wave number. Note that for amplitude coefficients,  $1 + r = t$ . For power coefficients,  $R + T = 1$ . These relations can be verified easily by substituting the above definitions of the reflection and transmission coefficients of the respective TE and TM polarizations. For normal incidence when  $\theta_i = \theta_t = 0$ ,  $r_{TE} = -r_{TM}$ , and  $R_{TE} = R_{TM}$ .

If the refractive index is complex, the reflection and transmission coefficients are also complex. In such cases, only the absolute value is taken into account.

The raytracer automatically computes the plane of incidence at each interface, decomposes the polarization vector into TE and TM components, and applies the respective reflection and transmission coefficients to these TE and TM components.

---

## Ray Photon Absorption and Optical Generation

When there is an imaginary component (extinction coefficient),  $\kappa$ , to the complex refractive index, absorption of photons occurs. To convert the absorption coefficient to the necessary units, the following formula is used for power/intensity absorption:

$$\alpha(\lambda) [\text{cm}^{-1}] = \frac{4\pi\kappa}{\lambda} \quad (730)$$

## Chapter 21: Optical Generation

### Raytracing

In the complex refractive index model, the refractive index is defined element-wise. In each element, the intensity of the ray is reduced by an exponential factor defined by  $\exp(-\alpha L)$  where  $L$  is the length of the ray in the element.

Therefore, the photon absorption rate in each element is:

$$G^{\text{opt}}(x, y, z, t) = I(x, y, z)[1 - e^{-\alpha L}] \quad (731)$$

$I(x, y, z)$  is the rate intensity (units of  $s^{-1}$ ) of the ray in the element. After all of the photon absorptions in the elements have been computed, the values are interpolated onto the neighboring vertices and are divided by its sustaining volume to obtain the final units of  $s^{-1} \cdot cm^{-3}$ . The absorption of photons occurs in all materials (including nonsemiconductor) with a positive extinction coefficient for raytracing. Depending on the quantum yield (see [Quantum Yield Models on page 658](#)), a fraction of this value is added to the carrier continuity equation as a generation rate so that correct accounting of particles is maintained.

---

## Using the Raytracer

The raytracer must be invoked within the unified interface for optical generation computation (see [Raytracing on page 671](#)) and can have the following options:

- Raytracing used in simple optical generation
- Monte Carlo raytracing
- Multithreading for raytracing
- Compact memory model for raytracing
- User-defined and distribution windows of starting rays
- Cylindrical coordinates for raytracing
- Different boundary conditions for raytracing
- Visualizing raytracing results

Optical generation by raytracing is activated by the `RayTracing` statement in the `Physics` section. An example of optical generation by raytracing is:

```
Plot {...
    RayTrees
}

Physics {...}
    Optics (
        ComplexRefractiveIndex(
            WavelengthDep(real imag)
            CarrierDep(real imag)
            GainDep(real) * or real(log)
```

## Chapter 21: Optical Generation

### Raytracing

```
    TemperatureDep(real)
    CRImodel (Name = "cristmodelname")
)
OpticalGeneration(...)
Excitation ...
    PolarizationAngle = 45          * deg
    Wavelength        = 0.5         * um
    Intensity          = 0.1         * W/cm2
    Window(
        Rectangle(dx = 1, dy = 2)
        IntensityDistribution(...)
    )
)
OpticalSolver (
    RayTracing(
        RayDistribution(...)
        MonteCarlo
        CompactMemoryOption      * reduced memory consumption
        MinIntensity = 1e-3       * fraction of start intensity
                                * to stop ray
        UseAverageMinIntensity * use average intensity of
                                * all starting rays as
                                * reference for MinIntensity
        DepthLimit = 1000        * maximum number of interfaces
                                * to pass
        RetraceCRIchange = 0.1   * fractional change of CRI
                                * to retrace
    )
)
)
}
```

The complex refractive index model must be used in conjunction with the raytracer. Various dependencies of the complex refractive index can be included, and they are controlled by parameters in the [ComplexRefractiveIndex](#) section of the parameter file (see [Complex Refractive Index Model on page 694](#)). The CRI model can be defined regionwise or materialwise.

The keyword `RetraceCRIchange` specifies the fractional change of the complex refractive index (either the real or imaginary part) from its previous state that will force a total recomputation of raytracing. To force retracing at every ramping point, you can set `RetraceCRIchange` to a negative number.

The starting rays are defined by a rectangular window or a user-defined window (see [Window of Starting Rays on page 709](#)).

---

## Terminating Raytracing

Raytracing offers different termination conditions:

- Raytracing is terminated if the ray intensity becomes less than  $n$  times (`MinIntensity` specifies  $n$ ) the original intensity of each starting ray.
- `DepthLimit` specifies the maximum number of material boundaries that the ray can pass through.
- `UseAverageMinIntensity` activates the computation of an average value of all the starting rays,  $I_{avg}$ , and raytracing terminates when the ray intensity becomes less than  $I_{avg} \times \text{MinIntensity}$ . This is only useful with the spatial intensity excitation profile (see [Spatial Intensity Function Excitation on page 684](#)).

---

## Monte Carlo Raytracing

In instances where rays are randomly scattered, for example on rough surfaces, a Monte Carlo–type raytracing is required, since you need to look at the aggregate solution of the raytracing process.

The concept of Monte Carlo raytracing follows that of the Monte Carlo method for carrier transport simulation. Suppose a ray impinges an interface. In the deterministic framework, the ray will split into a reflected part and a transmitted part at this interface. In the Monte Carlo framework, you track only one ray path and take the reflectivity as a probability constraint to decide if the ray is to be reflected or transmitted. As more rays impinge this material interface, the aggregate number of reflected rays will recover information about the reflectivity, and this is the crux of the Monte Carlo method. In a likewise manner, rough surface scattering gives an angular probability and, using the same strategy, the ensemble average of rays can model the physics of rough surface scattering.

As an example, the algorithm for the Monte Carlo raytracing at a regular material interface is:

- Compute the reflection and transmission coefficients,  $R$  and  $T$ , of the ray at the material interface.
- Generate a random number,  $r$ .
- If  $r \leq R$ , then choose to propagate the reflected ray only.
- If  $r > R$ , then choose to propagate the transmitted ray only.

In the case of special rays, such as those from the raytrace PMI, care must be taken to choose only a single propagating ray based on a new set of probabilistic rules.

## Chapter 21: Optical Generation

### Raytracing

The Monte Carlo raytracing has been implemented in both general raytracing and LED raytracing. The syntax is:

```
Physics {
    Optics (
        OpticalSolver (
            RayTracing (MonteCarlo)
        )
    )
}
```

#### Note:

Since only one ray path is chosen at each scattering event in the Monte Carlo method, the rays in the raytree will appear to have the same intensity values after scattering.

---

## Multithreading for Raytracer

Each raytree traced from the list of starting rays is mutually exclusive, so that the raytracer is an excellent candidate for parallelization.

To activate the multithreading capability of the raytracer, include the following syntax in the `Math` section of the command file:

```
Math {
    Number_Of_Threads = 2      # or maximum
    StackSize = 20000000      # increase to, for example, 20 MB
}
```

#### Note:

Raytracing is a recursive process, so it can easily obliterate the default (1 MB) stack space if the level of the raytree becomes increasingly deep. This might lead to unexplained segmentation faults or termination of the program. To resolve this issue, increase the stack space using the keyword `StackSize` in the `Math` section as shown in the syntax. The other solution is to use the compact memory model as described next.

---

## Compact Memory Model for Raytracer

The compact memory model for the raytracer does not store the raytree as it is being created, so it reduces the use of significant memory. Only essential information is tracked and stored, and this alleviates the amount of memory required. A reduced structure is used, and clever ways of extracting information in this reduced structure have been implemented without upsetting the multithreading feature of raytracing.

## Chapter 21: Optical Generation

### Raytracing

The compact memory model is activated as the default only for the unified interface for optical generation computation. The default is still switched off for the LED raytracing interface. A minus sign can be added in front of the keyword to deactivate it. If raytrees are requested to be plotted in the `Plot` statement, the compact memory model is switched off automatically, and a warning message appears in the log file to alert you of the changes. The command file syntax to switch off the compact memory option is:

```
Physics { ...
    Optics( OpticalSolver(...
        RayTracing ( -CompactMemoryOption )
    )
}
```

---

## Window of Starting Rays

Two mutually exclusive options for defining a set of starting rays are available: a user-defined set of rays or a distribution window. You can define these in the `Physics-Optics-OpticalSolver-Raytracing` statement syntax:

```
UserWindow (
    NumberOfRays = integer           # number of rays in file
    RaysFromFile = "filename.txt"    # position(um) direction area(cm^2)
    PolarizationVector = Random | ReadFromExcitation | ReadFromFile
)
RayDistribution (WindowName= "windowname1" ...)
```

#### Note:

The initial positions of starting rays must not be defined too close to or exactly on a vertex, an edge, a face, or an interface because the raytracer needs to determine the starting region of the rays.

## User-Defined Window of Rays

In the `UserWindow` section, you can enter your own set of starting rays using a text file. This means that you can choose the positions, directions, area, and polarization vector of each starting ray, thereby achieving greater flexibility.

The power (W) of each ray then is computed by multiplying the input area ( $\text{cm}^2$  in 3D or cylindrical simulations, and  $\text{cm}$  in 2D simulations) by the input wave power ( $\text{W}/\text{cm}^2$ ). Remarks are allowed but they must begin with a hash character (#). A sample of this file for 3D rays is:

```
filename.txt:
# Pos(x,y,z)[um]      Dir(x,y,z)[um]      Area[cm^2]  Polarization(x,y,z)
  0.0    0.0    0.0      0.0    0.0    1.0      1.0e-5     1  0  0
  0.0    0.05   0.0      0.0    0.0    1.0      1.0e-5     0  1  0
  0.0    0.05   0.05     0.0    0.0    1.0      1.0e-5     0  1  0
...
...
```

## Chapter 21: Optical Generation

### Raytracing

#### Note:

In 2D simulations, you must input the `Area` as a segment length (in cm). Consistent with 2D Sentaurus Device simulations, the third dimension is taken to be a default of 1  $\mu\text{m}$ , which means that the actual area for each user ray is the user `Area [cm]` multiplied by 1  $\mu\text{m}$ .

Different types of `PolarizationVector` can be chosen:

- `Random`: A random vector perpendicular to the ray direction is generated. In two dimensions, the generated random vector can be a 3D vector.
- `ReadFromFile`: The polarization vector is entered as the last three columns in the file.
- `ReadFromExcitation`: The polarization vector is constructed based on the information contained within the `Excitation` section. This is the default if the keyword `PolarizationVector` is omitted. The priority order for reading the polarization under this option is:

1. `OpticalSolver - RayTracing - PolarizationVector= (<float> <float> <float>)`
2. `OpticalSolver - RayTracing - PolarizationRandom`
3. `Excitation - PolarizationAngle= <float>`
4. `Excitation - Polarization= TE | TM | <float>`

If none of these options is found, `Excitation - PolarizationAngle=0` is used by default.

#### Note:

If `PolarizationVector` is set to `ReadFromExcitation` or `Random`, then the last three columns of the file that describe polarization are ignored.

In 2D simulations, the TE polarization vector is (0,0,1) and the TM polarization vector is formed by the cross product of the ray's direction vector and (0,0,1), with reference to [Figure 33 on page 703](#).

*Table 133 Requirements for the parameters in the input file for the user ray*

Parameter	2D simulation	Cylindrical simulation	3D simulation
Position [ $\mu\text{m}$ ]	(x,y,0)	(x,y,0)	(x,y,z)
Direction	(x,y,0)	(x,y,0)	(x,y,z)
Units of user input <code>Area</code>	cm (internally is multiplied by 1 $\mu\text{m}$ )	cm <sup>2</sup>	cm <sup>2</sup>
Polarization vector ( <code>ReadFromFile</code> only)	(x,y,z)	(x,y,z)	(x,y,z)

## Distribution Window of Rays

When the illumination window (see [Illumination Window on page 676](#)) is defined for raytracing, a corresponding `RayDistribution` section must be created to define how the excitation parameters can be translated in a raytracing context:

```
Physics {
    Optics (
        Excitation(
            Window ("windowname1")(
                Origin = (xx,yy)
                OriginAnchor = Center | North | South | East | West |
                    NorthEast | SouthEast | NorthWest | SouthWest
                RotationAngles = (phi, theta, psi)
                xDirection = (x,y,z)
                yDirection = (x,y,z)
                # You can define one of the following excitation shapes.
                # Rectangle, Circle, and Polygon are for three dimensions.
                Rectangle( ... )
                Line( ... )
                Circle( ... )
                Polygon( ... )
            )
            Window ("windowname2") (...)
            Window ("windowname3") (...)
        )
        OpticalSolver(
            Raytracing(
                RayDistribution(
                    WindowName = "windowname1"
                    Mode = Equidistant | MonteCarlo | AutoPopulate
                    NumberOfRays=<int> # for Mode=MonteCarlo | AutoPopulate
                    Dx=<float>           # for Mode=Equidistant
                    Dy=<float>           # for Mode=Equidistant
                    Scaling=<float>      # scaling factor
                )
                RayDistribution (WindowName="windowname2" ...)
                RayDistribution (WindowName="windowname3" ...)
            )
        )
    )
}
```

Multiple excitation windows and `RayDistribution` windows can be created. The only restriction is that every `RayDistribution` section must have a matching `Excitation` section. Matching is through the user-specified window name. If no window name for the `RayDistribution` section is specified, the parameters in that window are applied to all excitation windows, except those with matching window names.

The shape of the excitation is defined in the `Excitation(...Window(...))` section, and the shape can be a `Line` for two dimensions, and a `Rectangle`, `Circle`, or `Polygon` for three dimensions.

## Chapter 21: Optical Generation

### Raytracing

There are different ways in which you can create a set of starting rays on the excitation shape:

- Mode=Equidistant: The starting positions of the rays are arranged in a regular grid with intervals defined by `Dx` and `Dy`. Then, the grid is superimposed onto the excitation shape to extract an appropriate set of starting rays.
- Mode=MonteCarlo: Random positions are generated with the excitation shape to form the set of starting rays.
- Mode=AutoPopulate: A uniform grid of variable grid size is superimposed onto the excitation shape. The grid size is varied until the maximum possible number of grid points that is less than `NumberOfRays` can be fitted into the shape, and the set of starting ray positions is extracted from the fitted grid vertices.

---

## Cylindrical Coordinates for Raytracing

Body-of-revolution structures can be modeled optically in raytracing by tracing the rays in the 2D cut of the cylindrical structure and taking into account the reflection at the cylindrical axis and the ring volume of revolution.

To model body-of-revolution structures with raytracing:

1. Switch on the cylindrical coordinates in the `Math` section, and specify the location of the cylindrical axis. You can use either the keyword `xAxis` to specify the revolution around the `y`-axis (`xAxis=<float>`) or the keyword `yAxis` to specify the revolution around the `x`-axis (`yAxis=<float>`). The syntax is:

```
Math { ...
    Cylindrical(xAxis=<float>)      # default is xAxis=0
}
```

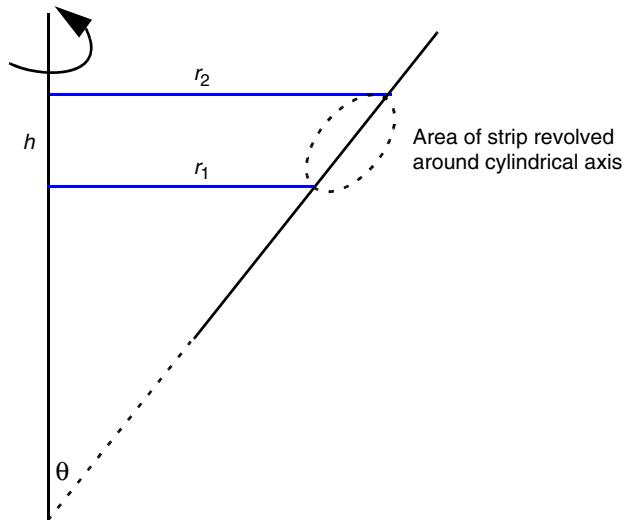
2. Define the illumination window appropriately, so that it lies on the same side as the device from the cylindrical axis, and both the window and device do not intersect the cylindrical axis.
3. There is no need to define any special reflecting boundary at the cylindrical axis. Sentaurus Device automatically creates an internal virtual reflecting boundary at the cylindrical axis so that any rays impinging it will be reflected.
4. The `RayDistribution` illumination window supports cylindrical coordinates. All methods of generating cylindrical starting rays are possible: `AutoPopulate`, `Equidistant`, and `MonteCarlo`. If the Gaussian spatial intensity is specified, the Gaussian shape is superimposed on the cylindrical ring-area weighted power of each starting ray, only for the `AutoPopulate` and `Equidistant` options. There is no support for the Gaussian spatial intensity and cylindrical `MonteCarlo` starting rays.

## Chapter 21: Optical Generation

### Raytracing

5. The illumination window can be at a skewed angle from the cylindrical axis, as shown in [Figure 34](#).

**Figure 34** An illumination window subtended at an angle  $\theta$  from the cylindrical axis, forming a conical surface of revolution



Assuming that the angle subtended between the illumination window and the cylindrical axis is  $\theta$ , the conical ring area of revolution is:

$$\text{Area} = \pi(r_1 + r_2)(r_2 - r_1) \frac{1}{\sin \theta} \quad (732)$$

This area is multiplied to each starting ray to retrieve the power carried by it. As a check, when  $\theta = 90^\circ$ , [Equation 732](#) reduces to that of a ring area.

---

## Boundary Condition for Raytracing

Special and spatially arbitrary boundary conditions can be specified in raytracing. There are two ways to define a boundary condition (BC) for raytracing by:

- Using special contacts.

In the contact-based definition, the boundaries are drawn as contacts (see [Specifying Electrical Boundary Conditions on page 115](#)) and are labeled accordingly using

Sentaurus Structure Editor. These contacts can be constructed specifically for setting a raytrace boundary condition, or they can coincide with an electrode or a thermode. To define clearly special raytrace boundary conditions, each line or surface contact defined must have a distinct name. This means that two parallel contacts must have different contact names, and intersecting contacts must also have different contact names. The contact is discretized into edges (2D) or faces (3D) after meshing.

## Chapter 21: Optical Generation

### Raytracing

- Using Physics MaterialInterface or Physics RegionInterface.

In the Physics Material Interface-based or the Physics RegionInterface-based definition, the boundaries are defined at the interface between the material or region. As with typical Sentaurus Device operation, RegionInterface takes precedence over MaterialInterface. After meshing, the interface is discretized into edges (two dimensions) or faces (three dimensions).

A mixture of contact-based and physics interface definitions of BCs is allowed. However, the contact-based definition takes precedence over the physics interface-based definition if there is an overlap.

Each edge or face defined as a special contact can only associate itself to one type of boundary condition. The following boundary conditions are listed in the order of preference, that is, if two boundary conditions are specified for the same edge or face, the higher ranked one is chosen:

1. Fresnel BC
2. Constant reflectivity/transmittivity BC
3. Raytrace PMI BC
4. Multilayer antireflective coating BC
5. Diffuse surface BC, implemented as an installed PMI
6. Periodic BC

## Fresnel Boundary Condition

The physics interface-based BC can quickly set many interfaces to a particular type of BC, especially if the material interface contains many region interfaces. What is missing is the disabling of a particular region interface from the general BC definition. Therefore, the Fresnel BC is introduced as a complement set to unset a particular region interface-based or contact-based BC. This can greatly simplify the command syntax input and enhance ease of use.

Furthermore, the Fresnel BC also can be used to sense the photon flux flowing through a particular interface when users activate the PlotInterfaceFlux feature (see [Plotting Interface Flux on page 728](#)).

The syntax for activating the Fresnel BC is:

```
RayTraceBC {  
    { Name= "contact1" Fresnel }  
}  
| Physics(RegionInterface="reg1/reg2") {  
    RaytraceBC (Fresnel)  
}
```

## Constant Reflectivity and Transmittivity Boundary Condition

In the command file, constant reflectivity and transmittivity boundaries must be specified by the following syntax:

```
RayTraceBC {
  { Name = "ref_contact1"
    Reflectivity = float
  }
  { Name = "ref_contact2"
    Reflectivity = float
    Transmittivity = float
  }
  ...
}
```

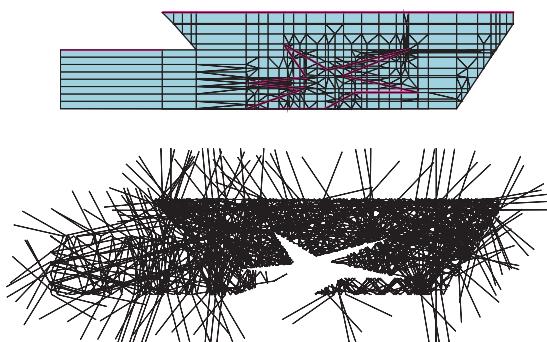
```
Physics (RegionInterface="regionname1/
regionname2") {
  RayTraceBC (
    Reflectivity = float
    Transmittivity = float
  )
}
```

These are power reflection,  $R$ , and transmission,  $T$ , coefficients. It is not necessary for  $R + T = 1$ . If  $R$  is specified only,  $T = 1 - R$ . If  $T$  is specified only,  $R = 1 - T$ .

For total absorbing or radiative boundary conditions, set `Reflectivity=0` and `Transmittivity=0` (or `Transmittivity=1`). Defining `Reflectivity=1` ensures that rays are totally reflected at that boundary. In the 2D case, reflection occurs at the edge of the element. In the 3D case, reflection occurs at the face of the element. This versatile boundary condition feature enables you to use symmetry to reduce the simulation domain.

Examples of the boundary condition whereby `Reflectivity` has been set to 1.0 are shown in [Figure 35](#) and [Figure 36](#). In the 2D case, a star boundary is drawn within a device; in the 3D case, a rectangular boundary is drawn within the device.

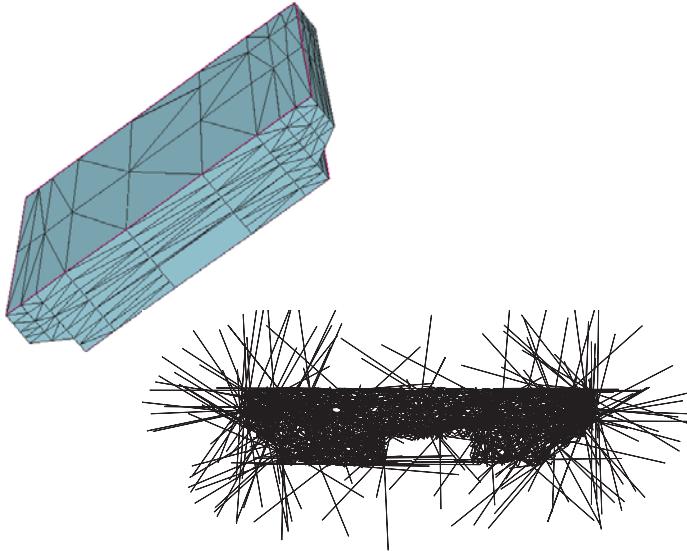
**Figure 35** Applying the reflecting boundary condition in a 2D LED simulation; the boundary is drawn as a star inside the device



## Chapter 21: Optical Generation

### Raytracing

Figure 36 Applying the reflecting boundary condition in a 3D LED simulation; the boundary is drawn as a hollow rectangular waveguide inside the device



## Raytrace PMI Boundary Condition

A special raytrace PMI BC can be defined. You can obtain useful information about the ray with this raytrace PMI and can modify some parameters of the ray. For details about this PMI and how it can be incorporated into the simulation, see [Preprocessing for Newton Iterations and Newton Step Control on page 1446](#).

As with the standard PMI, you can create a PMI section in the parameter file where you can initialize the parameters of the PMI. Note that the parameter must be specified either regionwise or material-wise in accordance to the standard PMI framework.

To activate the PMI, specify the location of the PMI BC and include the following syntax in the RayTraceBC section of the command file:

```
RayTraceBC { ...
  { Name = "pmi_contact"
    PMIModel = "pmi_modelname" }
}
Physics (MaterialInterface="materialname1/
           materialname2") {
  RayTraceBC (pmiModel = "modelname")
}
```

### Note:

Care must be taken to ensure that your PMI code is thread safe since the raytracing algorithm is multithreaded. Use only local variables and avoid global variables in your PMI code (see [Parallelization on page 1234](#)).

## Thin-Layer-Stack Boundary Condition

A thin-layer-stack boundary condition can be used to model interference effects in raytracing. The modeling of antireflective coatings used in solar cells is a typical example of the use of such boundary condition. The coatings are specified as special contacts and are treated as a boundary condition for the raytracer. The angle at which the ray is incident on the coating is passed as input to the TMM solver (the theory is described in [Transfer Matrix Method on page 734](#)), which returns the reflectance, transmittance, and absorbance for both parallel and perpendicular polarizations to the raytracer. The angle of refraction is calculated by the raytracer according to Snell's law, a direct implication of phase matching. This boundary condition is available for both 2D and 3D simulations.

The following command file excerpt, along with its demonstration in [Figure 37 on page 718](#), shows the use of this boundary condition:

```
RayTraceBC {  
    { Name="rayContact1"  
        reflectivity=1.0 }  
    { Name="rayContact2"  
        ReferenceMaterial = "Gas"  
        LayerStructure {  
            70e-3 "Nitride"; # um  
            6e-3 "Oxide"    # um  
        }  
    }  
}  
  
Physics (MaterialInterface="Gas/Silicon") {  
    RayTraceBC (  
        TMM (  
            ReferenceMaterial = "Gas"  
            LayerStructure {  
                70e-3 "Nitride"; # um  
                6e-3 "Oxide";   # um  
            }  
        )  
    )  
}
```

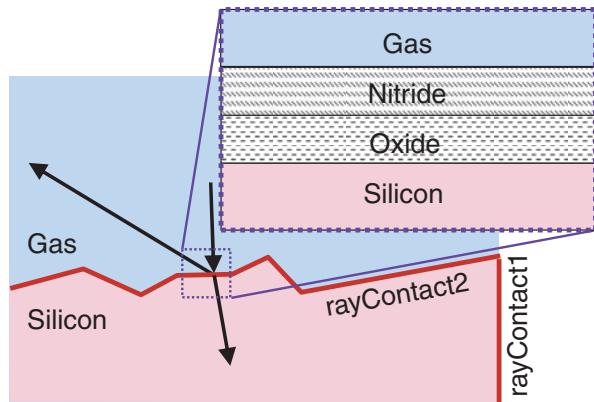
The first line in the `RayTraceBC` section above shows the definition of a constant reflectivity as a boundary condition (see [Fresnel Boundary Condition on page 714](#)). This option is usually chosen for the boundary of the simulation domain.

The other section defines a multilayer structure, for which the corresponding contact (`rayContact2`) in the grid file can be seen as a placeholder. Alternatively, you can use the `Physics` interface method of specifying such a special BC (as shown in the above syntax to the right). For each layer, the corresponding thickness [ $\mu\text{m}$ ] and material name must be specified. For the calculation of transmittance and reflectance, the transfer matrix method reads the complex refractive index from the respective parameter file. An additional parameter file or a new set of parameters must be added by you in either of the following cases: (a) a layer contains a material that does not exist in the grid file or (b) the material properties of a layer differ from the properties of a region in the grid file with the same material.

## Chapter 21: Optical Generation

### Raytracing

Figure 37 Illustration of thin-layer-stack boundary condition for simulation of an antireflection-coated solar cell



To fix the orientation of the multilayer structure with respect to the specified contact in the grid file, a `ReferenceMaterial` or a `ReferenceRegion` that is adjacent to the contact must be specified. The topmost entry of the `LayerStructure` table corresponds to the layer that is adjacent to the `ReferenceMaterial` or `ReferenceRegion`.

**Note:**

You can only specify a `ReferenceMaterial` or `ReferenceRegion` if the material names or region names, respectively, on either side of the contact *do not coincide*.

The multilayer structure is allowed to have an arbitrary number of layers.

## TMM Optical Generation in Raytracer

In modern thin-film solar-cell design, the multilayer thin film can be made of materials that can generate carriers by absorbing photons. To cater to such a phenomenon, the TMM contact in the raytracer has been modified to collect optical generations as rays traverse the TMM contact. The collected optical generations can then be distributed into specific regions of the electrical grid.

In addition, to model the correct optical geometry, the thin-film layers must be drawn into the device structure. However, the raytracer treats the physics of thin film using a TMM contact, and these thin layers should effectively be ignored during the raytracing process. As such, you need to use `VirtualRegions` (see [Virtual Regions in Raytracer on page 723](#)) to ignore these thin layers in the raytracing process.

## Chapter 21: Optical Generation

### Raytracing

The required syntax is:

```
RayTraceBC {...
  { Name= "TMMcontact"
    ReferenceMaterial= "Gas"
    LayerStructure {...}
    MapOptGenToRegions {
      "thinlayer1"
      "thinlayer2" ...
      QuantumEfficiency=float
    }
  }
}

Physics (RegionInterface="regionname1/
regionname2") {
  RayTraceBC (
    TMM (
      ReferenceMaterial = "Gas"
      LayerStructure {...}
      MapOptGenToRegions {"thinlayer1"
        "thinlayer2"...
      QuantumEfficiency = float
    )
  )
}

Physics {...
  Optics...
    OpticalSolver(
      RayTracing (...VirtualRegions{"toppml" "nitride" "oxide" ...})
    )
  )
}
}
```

A few comments about the syntax:

- `QuantumEfficiency` denotes the fraction of absorbed photons in the TMM BC that will be converted to optical generation. The keyword `QuantumYield` also can be specified in the unified raytracing interface. Therefore, the overall quantum yield of the TMM BC is a product of `QuantumEfficiency` and `QuantumYield`.
- The region names in the `MapOptGenToRegion` section refer to regions in the electrical device grid. The `MapOptGenToRegions{...}` list and `VirtualRegions{...}` list are independent, and there is no need for a one-to-one correspondence. For example, you can have `virtualRegions{r1,r2,r3}` and `MapOptGenToRegions{r2,r4}` whereby the entire optical generation from a TMM BC will be mapped onto regions `r2` and `r4` according to their volume ratio. The order of the region list is not relevant.
- For the optical generation mapping, a constant optical generation profile is assumed. The optical generation from a TMM BC is lumped into a constant value and distributed to different `MapOptGenToRegions` regions according to their volume ratios.
- The region names in the `VirtualRegions` section refer to regions in the optical device grid.

## Diffuse Surface Boundary Condition

Rough surfaces can be modeled by diffuse surface boundary conditions. The available diffuse BC models are:

- Phong scattering model with distribution:

$$I/I_0 = \cos^{\omega}(\alpha + \gamma) \quad (733)$$

where  $\omega$  is the order of the Phong model,  $\alpha$  is the scattered angle with respect to the scattering surface, and  $\gamma$  is an offset angle for the lobe of the distribution.

- Lambert scattering model with distribution:

$$I/I_0 = \cos(\alpha + \gamma) \quad (734)$$

**Note:**

The Lambert scattering model is a special case of the Phong model.

- Gaussian scattering model with distribution:

$$I/I_0 = \exp \frac{-(\alpha + \gamma)^2}{2\sigma^2} \quad (735)$$

where  $\sigma^2$  is the variance of the distribution.

- Random scattering model with uniform distribution

Within Sentaurus Device, a random number  $x \in [0, 1]$  is generated, and a mapping function is used to compute the scattering angle  $\alpha$ . The mapping function is computed as follows.

The random number  $x \in [0, 1]$  has a uniform distribution such that:

$$\int_0^1 f(x)dx = 1 \quad (736)$$

The scattering angle  $\alpha \in [0, \pi/2]$  has a distribution function,  $p(\alpha)$ , which can be the Phong, Lambert, or Gaussian model.

### 3D Case

The mapping requirement is:

$$p(\alpha)2\pi R \sin \alpha d\alpha = f(x)dx \quad (737)$$

and the normalization requirement is:

$$\int_0^{\pi/2} 2\pi R p(\alpha) \sin \alpha d\alpha = 1 \quad (738)$$

## Chapter 21: Optical Generation

Raytracing

### 2D Case

The mapping requirement is:

$$p(\alpha)Rd\alpha = f(x)dx \quad (739)$$

and the normalization requirement is:

$$\int_0^{\pi/2} p(\alpha)Rd\alpha = 1 \quad (740)$$

For the different scattering models,  $R$  must be derived based on the normalization conditions for 2D and 3D, respectively.

To find the inverse mapping of the random variable  $x$  to  $\alpha$ , the integration of the probability density functions to  $\alpha$  and  $x$  is performed, respectively:

$$\int_0^{\alpha} 2\pi R p(\alpha) \sin \alpha d\alpha = x \quad \text{for three dimensions} \quad (741)$$

$$\int_0^{\alpha} p(\alpha)Rd\alpha = x \quad \text{for two dimensions} \quad (742)$$

Then, the objective would be to express  $\alpha$  as a function of  $x$ . As an example, the inverse mapping function for the 3D Phong model is:

$$\alpha = \cos^{-1}[\omega + \sqrt{1-x}] \quad (743)$$

The inverse mapping functions of the other scattering models can be derived in a similar way.

The diffuse surface BC has been implemented as an installed PMI and can be invoked by including the following syntax in the Sentaurus Device command file:

```
RayTraceBC {
    { Name= "rough_contact"
        PMImodel= pmi_rtDiffuseBC( ... )
    }
}
```

<code>Physics(RegionInterface="region1/</code> <code>region2") {</code> <code>    RayTraceBC (</code> <code>        PMImodel=pmi_rtDiffuseBC ( ... )</code> <code>    )</code> <code>}</code>
--

where ... stands for the following scattering model parameters:

```
model = "Phong"          # "Lambert", "Random", "Gaussian"
phong_w = 200             # w parameter of Phong model
gaussian_sigma = 0.1      # sigma parameter of Gaussian model
set_randomseed = 123       # 0 to 10000, or -1=do not set
debug = 0                  # 1=print debug message, 0=do not print,
                           # 99999 = interactive debug
ReflectionTransmissionTable = "wavelengthRTtable.txt" # file input
```

## Chapter 21: Optical Generation

### Raytracing

```
pdfDimension = 2          # dimension of probability density function
reflectivity = 0.0        # reflectivity of rough surface
transmittivity = 1.0       # transmittivity of rough surface
```

Some comments about the parameter settings:

- You can set the specific reflectivity and transmittivity of the diffusive surface. However, if you still want to use the original reflectivity and transmittivity computed from the adjoining regions, set `reflectivity=-1` and `transmittivity=-1`.
- Setting a random seed allows for reproducibility of the simulation results.
- A useful `debug` option is included to ensure that the correct distribution is obtained.
- You can include the content of the "wavelengthRTtable.txt" file that contains a table of three columns: wavelength ( $\mu\text{m}$ ), reflectivity, and transmittivity. These are power reflectivity and transmittivity values, and enable the reflectivities and transmittivities of the specific diffuse boundary to change according to the wavelength.
- The `pdfDimension` refers to the dimension of the probability density function (PDF), and this can be chosen to be different from the dimension of the device. If this keyword is not included, the `pdfDimension` is set to the dimension of the device.

#### Note:

Instead of specifying the PMI parameters in the command file, you also can define them in the parameter file. To do so, in the command file, in the `RayTraceBC` section, set `PMIModel= "pmi_rtDiffuseBC"`. In the material parameter file, add a section `pmi_rtDiffuseBC{ ... }`, where `{ ... }` corresponds to the scattering model parameters as previously described.

## Periodic Boundary Condition

The periodic boundaries are limited to parallel X-, Y-, or Z-surfaces, so the device must have parallel surfaces in the direction of the periodicity. No special raytrace contacts need to be drawn onto the device, and you only need to specify the following syntax:

```
RayTraceBC {
    { Side="X" Periodic }
    { Side="Y" Periodic }
    { Side="Z" Periodic }
}
```

These periodic specifications are mutually exclusive, so you can specify a combination of any of them.

## Chapter 21: Optical Generation

### Raytracing

When the `PlotInterfaceFlux` feature (see [Plotting Interface Flux on page 728](#)) is activated, the corresponding entries describing the periodic fluxes are output to the plot file:

```
PeriodicFlux
  Xmin.FluxIn(region_name1)
  Xmin.FluxOut(region_name1)
  Xmax.FluxIn(region_name2)
  Xmax.FluxOut(region_name2)
  Ymin.FluxIn(region_name3)
  Ymin.FluxOut(region_name3)
  Ymax.FluxIn(region_name4)
  Ymax.FluxOut(region_name4)
  Zmin.FluxIn(region_name5)
  Zmin.FluxOut(region_name5)
  Zmax.FluxIn(region_name6)
  Zmax.FluxOut(region_name6)
```

#### Note:

Only those regions that contain any of the periodic BCs will be included in the list.

---

## Virtual Regions in Raytracer

This feature is a prelude to the TMM optical generation in the raytracer feature. Virtual regions can be defined in the raytracer such that rays ignore the presence of these regions during the raytracing process. In other words, when rays enter or leave a virtual region, no reflection or refraction occurs, and the ray is transmitted without change. This allows for additional flexibility in dual-grid simulations where some regions are important for electrical transport but insignificant for optics.

The syntax is:

```
Physics {...}
  Optics{...
    OpticalSolver(
      RayTracing ...
        VirtualRegions{"nitride" "oxide" "layer555" ...}
      )
    )
  }
}
```

---

## External Material in Raytracer

The default material surrounding a device is assumed to be air. In many cases, especially in LEDs, the device can be immersed in another material such as epoxy or some kind of phosphor. To correctly account for the directional, reflectivity, and transmittivity changes

## Chapter 21: Optical Generation

### Raytracing

caused by such external material, a user-defined file of wavelength-dependent complex refractive indices can be entered. The syntax is:

```
Physics {
    Optics (
        OpticalSolver (
            Raytracing (
                ExternalMaterialCRIFile = "string"
            )
        )
    )
}
```

The external material CRI file should contain three columns: wavelength ( $\mu\text{m}$ ), n, and k. A hash sign (#) is used to precede remarks. If the keyword `ExternalMaterialCRIFile` is not specified, the external medium is taken to be air with refractive index of 1.0.

---

## Additional Options for Raytracing

Additional options allow better control of raytracing in Sentaurus Device:

- Omitting reflected rays when performing raytracing
- Omitting weaker rays when performing raytracing

The syntax for these options is:

```
Physics { ...
    Optics(... ...
        OpticalSolver(
            RayTracing (
                OmitReflectedRays
                OmitWeakerRays
            )
        )
    )
}
```

---

## Redistributing Power of Stopped Rays

### Note:

This feature applies to LED raytracing. It does not work with Monte Carlo raytracing due to the fundamental assumption of the Monte Carlo method.

When raytracing terminates at a designated `DepthLimit` or `MinIntensity` value, there is still leftover power in those terminated rays. The sum of the powers contained in all these stopped rays can be redistributed into the raytree.

## Chapter 21: Optical Generation

### Raytracing

The total power of the rays is:

$$P_{\text{Total}} = P_{\text{abs}} + P_{\text{escape}} + P_{\text{stopped}} \quad (744)$$

where  $P_{\text{abs}}$  is the absorbed power,  $P_{\text{escape}}$  is the power of the escaped rays (out of the device), and  $P_{\text{stopped}}$  is the power of the rays terminated by the `DepthLimit` or `MinIntensity` condition.

Rearranging [Equation 744](#), the following expression is obtained:

$$P_{\text{Total}} = \frac{1}{1 - \frac{P_{\text{stopped}}}{P_{\text{Total}}}} (P_{\text{abs}} + P_{\text{escape}}) \quad (745)$$

Therefore, by multiplying a redistribution factor to  $P_{\text{abs}}$  and  $P_{\text{escape}}$ , the leftover power in the stopped rays can be accounted for. The syntax is:

```
Physics { ...
    Optics(...
        OpticalSolver(
            RayTracing (... 
                RedistributeStoppedRays
            )
        )
    )
}
```

---

## Weighted Interpolation for Raytrace Optical Generation

In raytracing, first optical absorption is computed elementwise, and then it is distributed evenly onto the associated vertices of the element. However, as a ray traverses an element, it can be closer to one particular corner or vertex of the element, in which case, distributing the optical absorption evenly onto all vertices of the element does not give a true picture. A better approximation is to use a weighted interpolation approach to distribute the optical absorption to vertices that are nearer to the center of each ray that traverses the element. The ray center is defined as the midpoint of the ray with respect to the optical absorption. This will improve the representation and accuracy of optical absorption profiles, and reduce the dependency on mesh size. The syntax is:

```
Physics{
    Optics(
        OpticalSolver(
            Raytracing( WeightedOpticalGeneration )
        )
    )
}
```

---

## Visualizing Raytracing

The keyword `RayTrees` can be set in the `Plot` section to visualize raytracing using the TDR format. Thereby, an additional geometry is added to the plot file representing the ray paths. The following datasets are available for each ray element:

- `Depth`: Number of material boundaries that the ray has passed through
- `Intensity`: Ray intensity
- `Transmitted` (Boolean): True, as long as the ray is transmitted

**Note:**

`RayTrees` cannot be plotted with the compact memory option.

---

## Computing Optical Intensity

Due to the nature of raytracing, the spatial intensity distribution  $I$  is not well defined. However, it can be computed from its relation to the APD as:

$$I = E_{\text{ph}} \frac{\text{APD}}{\alpha}$$

where  $E_{\text{ph}}$  is the photon energy and  $\alpha$  is the optical absorption coefficient. However, the relation fails for nonabsorbing regions where  $\alpha = 0$ .

To resolve this, a virtual APD concept can be used for nonabsorbing regions, where the absorption coefficient is set to a small value,  $\varepsilon$ , for the purposes of computing a virtual APD by using:

$$\text{vAPD} = \frac{P}{E_{\text{ph}}} \frac{(1 - e^{-\varepsilon \cdot d})}{V}$$

where  $P$  is the power deposited in an element with volume  $V$  and  $d$  is the propagation distance in the element. The intensity can then be computed by using:

$$I = E_{\text{ph}} \frac{\text{vAPD}}{\varepsilon}$$

For absorbing regions with a nonzero absorption coefficient,  $\alpha$ , the APD can be computed as:

$$\text{APD} = \frac{P}{E_{\text{ph}}} \frac{(1 - e^{-\alpha \cdot d})}{V}$$

Then, the intensity can be computed by using  $I = E_{\text{ph}} \frac{\text{APD}}{\alpha}$ .

## Chapter 21: Optical Generation

### Raytracing

The parameter `IntensityAbsCoeff` is used to set the value of  $\alpha$  (the default is 1e-5):

```
Physics {
    Optics(
        OpticalSolver(
            RayTracing( ... IntensityAbsCoeff=<float> )
        )
    )
}
```

---

## Reporting Various Powers in Raytracing

Various powers are reported in the log file after a raytracing event, and the format is as follows:

```
Summary of RayTrace Total Photons and Powers:
      Input   Escaped   StoppedMinInt   StoppedDepth   AbsorbedBulk   AbsorbedBC
Photons [#/s]: 4.531E+11  1.333E+11  4.142E+07  0.000E+00  6.236E+10  2.574E+11
Powers [W]:   3.000E-07  8.826E-08  2.743E-11  0.000E+00  4.129E-08  1.704E-07
```

A brief summary of these powers is:

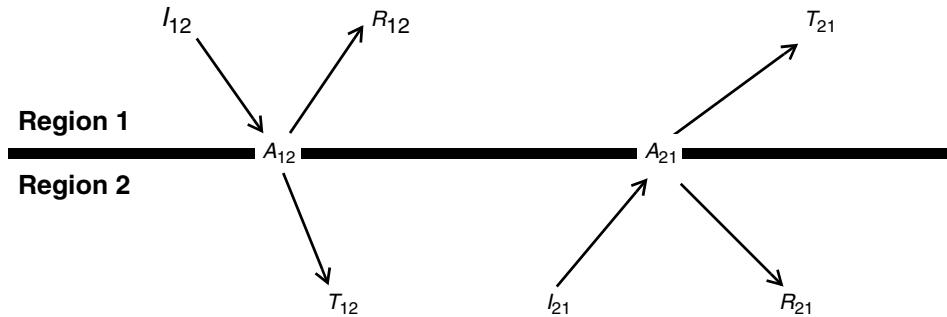
- Input power is computed by multiplying the input wave power (units of  $\text{W}/\text{cm}^2$ ) by the rectangular or circular window area where the starting rays have been defined.
- In two dimensions, the length in the third dimension is taken as 1  $\mu\text{m}$ , conforming to the rest of the Sentaurus Device 2D treatment. The total input power listed should correspond to the `Intensity` in the `Excitation` section multiplied by the actual area of the starting ray window. If an area factor has been defined in the simulation, this area factor also is multiplied into the result.
- Escaped power refers to the sum of powers of the rays that are ejected from the device and are unable to re-enter the device.
- StoppedMinInt power sums the powers of the rays terminated by the `MinIntensity` condition.
- StoppedDepth power sums the powers of the rays terminated by the `DepthLimit` criteria.
- AbsorbedBulk power refers to power absorbed in bulk regions.
- AbsorbedBC power refers to power absorbed in the TMM contacts. This column is shown only if TMM contacts have been defined.

In the plot file, the raytrace photon rates and powers are listed under `RaytracePhoton` and `RaytracePower`, respectively. In addition, the ratio of the various powers and the input power are plotted under `RaytraceFraction`.

## Plotting Interface Flux

The photon flux flowing through any interface can be tracked with the `PlotInterfaceFlux` feature. The tracking is made possible by recording the reflected, transmitted, and absorbed flux flowing through each interface or contact BC. Since photon flux is directional, there is a need to distinguish between the flux flowing from Region 1 to Region 2, or from Region 2 to Region 1, as shown in [Figure 38](#).

*Figure 38 Distinguishing directional photon flux*



Consider the photon fluxes (carried by rays) flowing from Region 1 to Region 2. For photon flux conservation:

$$I_{12} = R_{12} + T_{12} + A_{12} \quad (746)$$

where:

- $I_{12}$  is the incident flux.
- $R_{12}$  is the reflected flux.
- $T_{12}$  is the transmitted flux.
- $A_{12}$  is the flux absorbed at the interface.

The reverse is true for flux flowing from Region 2 to Region 1. At each interface, the summation of  $R_{12}$ ,  $T_{12}$ ,  $A_{12}$ ,  $R_{21}$ ,  $T_{21}$  and  $A_{21}$  is stored in the raytracing process.

With this information, you can compute various reflection and transmission coefficients at the interface that has been declared as a raytrace BC. For example, assume that light is illuminated from Region 1 to Region 2. Within Region 2, there can be multiple reflections, and some rays might escape back into Region 1 through  $T_{21}$ .

Therefore, the power reflection coefficient for the case in [Figure 38](#) is:

$$\tilde{R}_{12} = \frac{R_{12} + T_{21}}{R_{12} + T_{12} + A_{12}} \quad (747)$$

## Chapter 21: Optical Generation

### Raytracing

The following syntax activates the feature for tracking and plotting interface fluxes:

```
Physics { ...
    Optics (... 
        OpticalSolver (... 
            RayTracing ( PlotInterfaceFlux )
        )
    )
}
```

This feature is limited to the unified raytracing interface. In the plot file, only the fluxes of the interfaces or contacts that have been declared as a raytrace BC are output. In addition, activating the `PlotInterfaceFlux` feature also enables the output of the absorbed photon density in every layer of the TMM BC. The output variables in the plot file are listed as follows:

```
RaytraceInterfaceFlux          # directional
    R(region1/region3)
    T(region1/region3)
    A(region1/region3)
    R(region3/region1)
    T(region3/region1)
    A(region3/region1)

RaytraceContactFlux           # directional
    R(contactname1(region1/region3))
    T(contactname1(region1/region3))
    A(contactname1(region1/region3))
    R(contactname1(region3/region1))
    T(contactname1(region3/region1))
    A(contactname1(region3/region1))

RaytraceInterfaceTMLayerFlux   # not directional
    A(region1/region3).layer1
    A(region1/region3).layer2
    A(region1/region3).layer3

RaytraceContactTMLayerFlux    # not directional
    A(contactname1(region1/region3)).layer1
    A(contactname1(region1/region3)).layer2
    A(contactname1(region1/region3)).layer3
```

A contact-based BC can possibly intersect many region interfaces. In the plot file output, only a representative region interface is used to indicate the directional flow of photon flux for the entire contact BC.

When the `PlotInterfaceFlux` feature is activated, the `OpticalIntensity` computation will change such that the intensity within each region is scaled to the net photon flux gained in that region. Therefore, the `OpticalIntensity` values can be negative in some regions. Integration of the `OpticalIntensity` within the region recovers the net photon flux flowing into that region.

## Far Field and Sensors for Raytracing

To collect information on the rays that exit a device, the concept of far field and sensors is introduced in the unified raytracer interface.

The far field is collected on a virtual circle for two dimensions and a virtual sphere for three dimensions. Ray collection does not destroy the rays. On the virtual far-field surface, the far-field intensity is computed based on a unit measure of radius, whereby the collected ray powers at each interval are divided by the radian angular arc (2D) or area (3D).

$$\text{In two dimensions, far-field intensity} = \frac{\text{raypowers}}{d\phi}.$$

$$\text{In three dimensions, far-field intensity} = \frac{\text{raypowers}}{\{\cos((\theta_1) - \cos(\theta_2))\}d\phi}.$$

On the other hand, a sensor sums the total power of all rays impinging the sensor. Two types of sensor can be defined:

- A line segment sensor in two dimensions or a flat rectangular sensor in three dimensions.
- An angular sensor that collects the power of the rays that radiate within the angular span of the sensor. In two dimensions, a range of  $\phi$  defines the sensor; whereas in three dimensions, ranges of  $\theta$  and  $\phi$  are needed. These are defined according to the regular Cartesian coordinate system. In two dimensions,  $\phi$  rotates from 0 at the positive x-axis in a counterclockwise direction towards the positive y-axis. In three dimensions,  $\theta$  rotates from 0 at the positive z-axis towards the xy plane; whereas,  $\phi$  is similarly defined as in two dimensions.

The `SensorSweep` option is also available for 3D simulations. It allows you to visualize the variation of ray power collected on a latitude or longitude ring band.

The syntax for activating the far field and sensors in the unified interface for optical generation computation is:

```

Physics {
    Optics (
        OpticalSolver (
            Raytracing(
                Farfield(
                    Origin = auto | <vector>           # default is auto
                    Discretization = <integer>         # default is 360 for 2D,
                                                # 36 for 3D
                    ObservationRadius = <float>       # default is 1e6 um
                                                # (1 meter)
                Sensor(
                    Name = "sensornamel"
                    Rectangle (                      # 3D only
                        Corner1 = <vector>

```

## Chapter 21: Optical Generation

### Raytracing

```
        Corner2 = <vector>
        Corner3 = <vector>
        UseNormalFlux
        AxisAligned           # 3D only
    }
    Line (                           # 2D only
        Corner1 = <vector>
        Corner2 = <vector>
        UseNormalFlux
    )
    Angular (
        Theta = (<float> <float>)
        Phi = (<float> <float>)
    )
)
Sensor(
    Name = "sensorname2"
    Rectangle ( ... )          # 3D only
    Line ( ... )                # 2D only
    Angular ( ... )
)
SensorSweep(                      # 3D only
    Name = "sensorname3"
    Ndivisions = <integer>
    VaryPhi
    Theta = (<float> <float>) # Use with VaryPhi
)
SensorSweep(                      # 3D only
    Name = "sensorname4"
    Ndivisions = <integer>
    VaryTheta
    Phi = (<float> <float>)   # Use with VaryTheta
)
)
)
)
)
}
}
```

Some comments about the syntax:

- Multiple **far-field Sensor** and **SensorSweep** sections can be defined. However, each **Sensor** and **SensorSweep** section must have a unique name for identification.
- When **Line** or **Rectangle** sensors are used, the option **UseNormalFlux** allows you to sum the normal-projected power from the ray that is impinging the sensor at an angle.
- In the case of a **Rectangle** sensor in three dimensions, if you specify **AxisAligned**, then only opposing corners of the rectangle are required. Sentaurus Device automatically computes the other two corners of the rectangle.

## Chapter 21: Optical Generation

### Raytracing

- If an empty Farfield() section is specified, then the far field is plotted with the default values of Discretization, Origin, and ObservationRadius.
- The far field takes values of intensity type. Therefore, to recover the total number of photons, you must integrate over the angle (in radian).
- The 3D far field is projected onto a staggered grid (see [Staggered 3D Grid LED Radiation Pattern on page 1063](#)).
- The far field is plotted at every instance where the Plot statement in the Solve section is activated. The file names of the far field and sensor sweep are derived from the plot file name. For example, this syntax:

```
File { ...
    Plot = "n99_des"
}
Solve { ...
    Plot ( Range=(0,1) Intervals=5 )
}
```

produces the following far-field files:

```
n99_000000_des_farfield.tdr
n99_000001_des_farfield.tdr
...
n99_000000_des_sensorsweep.plt
n99_000001_des_sensorsweep.plt
...
```

- In the SensorSweep plot file, all the user-defined SensorSweep sections are defined in the following fields of the plot file:

```
sensorname3
    Phi
    Photons_DelTheta
sensorname4
    Theta
    Photons_DelPhi
```

- The values of the sensor are added to the general plot file of the simulation, and the field entries are:

```
RaytraceSensor
    sensorname1
    sensorname2
```

---

## Dual-Grid Setup for Raytracing

Raytracing in Sentaurus Device is based on tracing the rays in each cell of the simulation mesh. For simulations in which the optical material properties do not vary on a short length scale, this might lead to the unnecessary deterioration of simulation performance. In such

## Chapter 21: Optical Generation

### Raytracing

cases, a dual-grid setup can be used, which allows the use of a coarse mesh for raytracing, while the electronic equations are solved on a finer mesh (see [Figure 39 on page 734](#)).

The basic syntax for setting up a raytracing dual-grid simulation is:

```
File {
    Output  = "output"
    Current = "current"
}
Plot {
    OpticalIntensity
    OpticalGeneration
}
# Specify grid and material parameter file names for raytracing:
OpticalDevice OptGrid {
    File {
        Grid      = "raytrace.tdr"
        Doping    = "raytrace.tdr"
        Current   = "opto"
        Plot      = "opto"
        Parameter = "CIS.par"
    }
    Physics { HeteroInterface }
}
# Specify grid, material parameters, and physical models for
# electrical simulation:
Device CIS {
    Electrode {
        { Name="sub" Voltage = 0.0 }
        { Name="pd"  Voltage = 0.0 }
    }
    File {
        Grid      = "in_elec_pof.tdr"
        Doping    = "in_elec_pof.tdr"
        Parameter = "CIS.par"
        Current   = "current"
        Plot      = "plot"
    }
    Physics {
        AreaFactor = 1
        Optics(... {
            ComplexRefractiveIndex(...)
            WavelengthDep(real imag)
        })
        OpticalGeneration(...)
        Excitation(...)
        OpticalSolver(
            RayTracing (
                RayDistribution(...)
                DepthLimit = 1000
                MinIntensity = 1e-3
            )
        )
    }
}
```

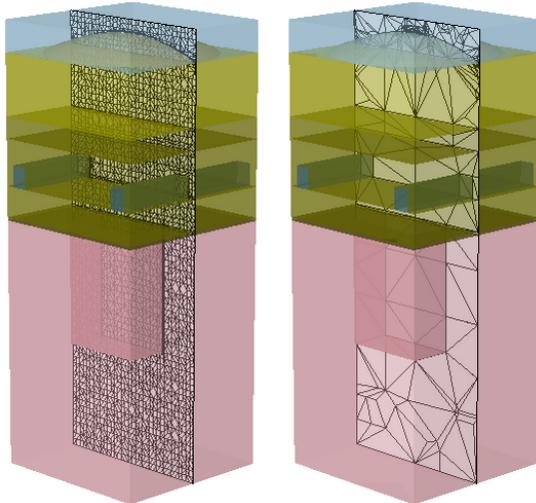
## Chapter 21: Optical Generation

### Transfer Matrix Method

```
)  
}  
# Specify system connectivity:  
System {  
    OptGrid opt ()  
    CIS d1 (pd=vdd sub=0) { Physics{ OptSolver = "opt" } }  
    Vsource_pset drive(vdd 0){ dc = 0.0 }  
}  
}  
  
Solve { Poisson }
```

The dual-grid simulation setup also allows the unified raytracing interface to be used (see [Raytracing on page 671](#)).

**Figure 39** Comparison of two grids used in a 3D dual-grid CMOS image sensor simulation; the figures show cuts through the device center; (left) the electronic equations are solved on a fine grid and (right) a coarse grid is used for raytracing



---

## Transfer Matrix Method

Sentaurus Device can calculate the propagation of plane waves through layered media by using a transfer matrix approach.

---

## Physical Model

In the underlying model of the optical carrier generation rate, monochromatic plane waves with arbitrary angles of incidence and polarization states penetrating a number of planar, parallel layers are assumed. Each layer must be homogeneous, isotropic, and optically

## Chapter 21: Optical Generation

### Transfer Matrix Method

linear. In this case, the amplitudes of forward and backward running waves  $A_j^\pm$  and  $B_j^\pm$  in each layer in [Figure 40](#) are calculated with help of transfer matrices.

These matrices are functions of the complex wave impedances  $Z_j$  given by  $Z_j = n_j \cdot \cos \Theta_j$  in the case of E polarization (TE) and by  $Z_j = n_j / (\cos \Theta_j)$  in the case of H polarization (TM). Here,  $n_j$  denotes the complex index of refraction and  $\Theta_j$  is the complex counterpart of the angle of refraction ( $n_0 \cdot \sin \Theta_0 = n_j \cdot \sin \Theta_j$ ).

The transfer matrix of the interface between layers  $j$  and  $j+1$  is defined by:

$$T_{j,j+1} = \frac{1}{2Z_j} \begin{bmatrix} Z_j + Z_{j+1} & Z_j - Z_{j+1} \\ Z_j - Z_{j+1} & Z_j + Z_{j+1} \end{bmatrix} \quad (748)$$

The propagation of the plane waves through layer  $j$  can be described by the transfer matrix:

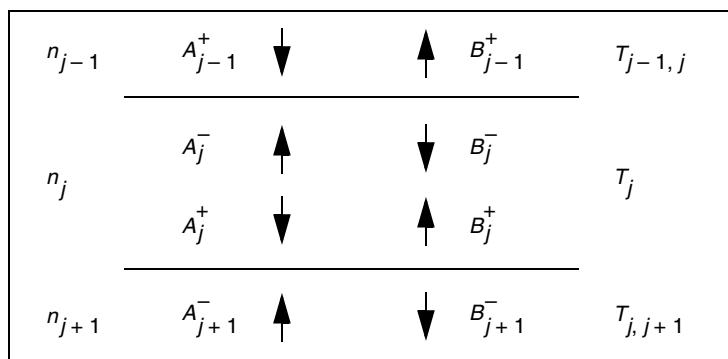
$$T_j(d_j) = \begin{bmatrix} \exp 2\pi i n_j \cos \Theta_j \frac{d_j}{\lambda} & 0 \\ 0 & \exp -2\pi i n_j \cos \frac{\Theta_j d_j}{\lambda} \end{bmatrix} \quad (749)$$

with the thickness  $d_j$  of layer  $j$  and the wavelength  $\lambda$  of the incident light.

The transfer matrices connect the amplitudes of [Figure 40](#) as follows:

$$\begin{aligned} \begin{matrix} B_j^+ \\ A_j^+ \end{matrix} &= T_{j,j+1} \cdot \begin{matrix} A_{j+1}^- \\ B_{j+1}^- \end{matrix} \\ \begin{matrix} A_j^- \\ B_j^- \end{matrix} &= T_j(d_j) \cdot \begin{matrix} B_j^+ \\ A_j^+ \end{matrix} \end{aligned} \quad (750)$$

*Figure 40 Wave amplitudes in a layered medium and transfer matrices connecting them*



## Chapter 21: Optical Generation

### Transfer Matrix Method

It is assumed that there is no backward-running wave behind the layered medium, and the intensity of the incident radiation is known. Therefore, the amplitudes  $A_j^\pm$  and  $B_j^\pm$  at each interface can be calculated with appropriate products of transfer matrices.

For both cases of polarization, the intensity in layer  $j$  at a distance  $d$  from the upper interface ( $j, j+1$ ) is given by:

$$I_{T(\text{TE, TM})}(d) = \frac{\Re(Z_j)}{\Re(Z_0)} \cdot \left\| T_j(d) \cdot \begin{array}{c} A_j^- \\ B_j^- \end{array} \right\|^2 \quad (751)$$

with the proper wave impedances. If  $\delta$  is the angle between the vector of the electric field and the plane of incidence, the intensities have to be added according to:

$$I(d) = I_{\text{TM}}(d) + I_{\text{TE}}(d) \quad (752)$$

where  $I_{\text{TM}} = (1 - a)I(d)$  and  $I_{\text{TE}} = aI(d)$  with  $a = \cos^2 \delta$ .

One of the layers must be the electrical active silicon layer where the optical charge carrier generation rate  $G^{\text{opt}}$  is calculated. The rate is proportional to the photon flux  $\Phi(d) = I(d)/\hbar\omega$ .

In the visible and ultraviolet region, the photon energy  $\hbar\omega$  is greater than the band gap of silicon. In this region, the absorption of photons by excitation of electrons from the valence to the conduction band is the dominant absorption process for nondegenerate semiconductors. Far from the absorption threshold, the absorption is considered to be independent of the free carrier densities and doping. Therefore, the silicon layer is considered to be a homogeneous region.

The absorption coefficient  $\alpha$  is the relative rate of decrease in light intensity along its path of propagation due to absorption. This decrease must be distinguished from variations caused by the superposition of waves. Therefore, the rate of generated electron–hole pairs is:

$$G_0^{\text{opt}} = \alpha\eta \frac{I(d)}{\hbar\omega} \quad (753)$$

where the absorption coefficient  $\alpha$  is given by the imaginary part of  $4\pi Z_{\text{Si}}/\lambda$ . The quantum yield  $\eta$  is defined as the number of carrier pairs generated by one photon. More details about the available quantum yield models and their specification in the `OpticalGeneration` section of the command file are given in [Quantum Yield Models on page 658](#).

## Current Plot Quantities

The quantities reflection (R), transmission (T), and absorption (A) as well as the phase shift for the reflected wave ( $R_{\text{phase}}$ ) and for the transmitted wave ( $T_{\text{phase}}$ ) are computed and added to the current plot file under the dataset group `LayerStack(<windowname>)`, where `<windowname>` is the name of the TMM window and defaults to `unnamed_0`. All quantities are

computed for TE and TM polarization. In addition to R, T, A, the total quantity corresponding to the mean value of TE and TM is plotted.

## Rough Surface Scattering

To model the effect of light trapping due to scattering of incident light at rough interfaces of a planar multilayer structure, the standard TMM approach describing the propagation of coherent light is extended. Part of the coherent light incident at a rough interface is scattered and spreads incoherently throughout the structure. Scalar scattering theory [4][5] allows to approximate the amount of scattered light at a rough interface. The so-called haze parameter defines the ratio of diffused (scattered) light to total (diffused + specular) light. The directional dependency of the scattering process is modeled by angular distribution functions (ADFs). In this scenario, a rough interface is characterized by its haze function (haze parameter as a function of the wavelength of incident light) for reflection and transmission as well as the ADF for direct coherent light incident at the interface and the ADF for scattered light incident at the interface.

The implementation of rough surface scattering within the existing TMM framework follows the semi-coherent optical model outlined in [6]. It is based on the following assumptions:

- The haze factor determines the fraction of direct light, transferred to scattered light.
- Coherent and diffused light incident on a rough interface are scattered differently using different ADFs.
- Interfaces are nonabsorptive, meaning photon flux is conserved at interfaces.
- For scattered light incident at a rough interface, the mean value of TE and TM polarized light for reflectance and transmittance is used.
- For the scattered light, the total reflectance and transmittance for a rough interface are assumed to be equal to the ones for the corresponding flat interface.
- The phase change of the electric field traveling across a rough interface is the same as in the case of the corresponding flat interface.
- Scattered light incident at a rough interface is further split, according to its haze factor, into a specular part, which corresponds to light incident at a flat interface, and a diffused part leading to further spreading of the incident scattered light.

Given these assumptions, the specular and diffused components of light at each interface can be expressed in terms of the corresponding haze functions and ADFs. The description of coherent light incident at a rough interface differs from the approach taken in [6] in that a generalized interface matrix has been derived. It accounts for the reduced reflection and transmission according to the respective haze functions and allows to keep the common TMM computation approach for the electric field and intensity.

## Chapter 21: Optical Generation

### Transfer Matrix Method

The reduced interface matrix reads as follows:

$$\tilde{T}_{j,j+1} = \frac{1}{h_{j+1,j}^t} \begin{bmatrix} 1 & (h_{j+1,j}^r r_{j,j+1}) \\ (h_{j,j+1}^r r_{j,j+1}) & (h_{j,j+1}^t h_{j+1,j}^t + (h_{j+1,j}^r h_{j,j+1}^r - h_{j,j+1}^t h_{j+1,j}^t) r_{j,j+1}^2) \end{bmatrix} \quad (754)$$

where  $r_{j,j+1}$  and  $t_{j,j+1}$  denote the Fresnel reflection and transmission coefficients for normal incidence given by:

$$r_{j,j+1} = \frac{n_j - n_{j+1}}{n_j + n_{j+1}} \quad (755)$$

$$t_{j,j+1} = \frac{2n_j}{n_j + n_{j+1}} \quad (756)$$

and  $h_{j,j+1}^r$  and  $h_{j,j+1}^t$  are reduction factors determined by the haze functions for reflection and transmission:

$$h_{j,j+1}^r = \sqrt{1 - H_{j,j+1}^r} \quad (757)$$

$$h_{j,j+1}^t = \sqrt{1 - H_{j,j+1}^t} \quad (758)$$

The haze functions for reflection and transmission derived from the scalar scattering theory are:

$$H_{j,j+1}^r(\lambda, \phi_j) = 1 - \exp\left[-\frac{4\pi\sigma_{\text{rms}} c_r(\lambda, \sigma_{\text{rms}}) n_j \cos \phi_j}{\lambda} a^r\right] \quad (759)$$

$$H_{j,j+1}^t(\lambda, \phi_j) = 1 - \exp\left[-\frac{4\pi\sigma_{\text{rms}} c_t(\lambda, \sigma_{\text{rms}}) |n_j \cos \phi_j - n_{j+1} \cos \phi_{j+1}|}{\lambda} a^t\right] \quad (760)$$

where  $\sigma_{\text{rms}}$  stands for the root-mean-square roughness of the interface and  $\lambda$  is the wavelength of the incident light. The exponent  $a^{r/t}$  and the correction function  $c_{r/t}$  are fitting parameters to better match experimental data.

#### Note:

The haze functions defined in [Equation 759](#) and [Equation 760](#) are given in their general form, which also covers light incident at the angle  $\phi_j$  from the surface normal. This expression is needed in the description of scattered light incident at a rough interface.

For the propagation of the angular intensity components of scattered light through each layer, the effective path length is used to stay within the one-dimensional formalism:

$$I(\lambda, \phi_j, d) = I(\lambda, \phi_j) \exp \left[ -\frac{\alpha(\lambda)d}{\cos \phi_j} \right] \quad (761)$$

The scattering solver first propagates all forward-going components through the layer structure and completes a round trip by propagating all backward-going components. In

## Chapter 21: Optical Generation

### Transfer Matrix Method

each round trip, part of the light might be absorbed within the structure as well as transmitted through the front or back side, depending on the material properties at the given wavelength. This leads to a reduction of the light intensity components at the various interfaces with increasing number of iterations. When the ratio of the largest remaining intensity component in the system to the sum of all initial intensity components falls below the value of `Tolerance`, the solver terminates. As a consequence, the residual intensity of each component at all interfaces is not included in the extracted results.

The overall simulation flow consists of the following steps:

- Solving the coherent propagation problem following the default TMM solver algorithm, but using the generalized interface matrices for rough interfaces. In this step, the starting light intensity for scattered light is calculated at each interface as well.
- The scattered light is propagated through the structure following an iterative approach similar to raytracing as outlined in [6]. However, other than raytracing, instead of single rays, ray bundles with discrete angular distribution are propagated from layer to layer. The number of iterations needed to solve the scattering problem to the required accuracy mainly depends on how much light is absorbed within the structure and how much light is coupled out at the front and back ends during a single round trip.
- The scattered and coherent absorbed photon density profiles are summed and passed to the quantum yield model.

#### Note:

Support for rough surface scattering is limited to normal incident light.

For more information on how to apply the rough surface scattering model and its configuration options, see [Using Scattering Solver on page 741](#).

---

## Using Transfer Matrix Method

The transfer matrix method is only supported by the unified interface for optical generation computation (see [Overview of Optical Generation on page 648](#)). The `OpticalGeneration` section activates the computation of the optical generation along with optional parameters such as the quantum yield. The `OpticalSolver` section determines the underlying optical solver together with solver-specific parameters as shown in the following example:

```
Physics {...
  Optics ...
    OpticalGeneration (...)

    ComplexRefractiveIndex (...)

    Excitation ...
      Window ("L1") ...
    )

    OpticalSolver (
      TMM (
```

## Chapter 21: Optical Generation

### Transfer Matrix Method

```
PropagationDirection = Perpendicular # Default Refractive
NodesPerWavelength = 20
LayerStackExtraction (...
    WindowName = "L1"
    Medium (...))
)
)
)
Excitation (...  
    Window ("L1") (...))
)
}
}
```

The properties of the incident light such as `Wavelength`, `Intensity`, `Polarization`, and `angle of incidence Theta` are specified in the `Excitation` section (see [Setting the Excitation Parameters on page 674](#)) along with one or several illumination windows (see [Illumination Window on page 676](#)), which confine the incident light to a certain part of the device structure.

An illumination window determines how the 1D solution of the transfer matrix method is interpolated to the higher dimensional device grid. The minimum coordinate of the 1D profile is pinned to the illumination window, and the interpolation of the profile in propagation direction can be performed either perpendicular to it or according to Snell's law, which is the default. The corresponding keyword in the `TMM` section is `PropagationDirection`, which can take either `Perpendicular` or `Refractive` as its argument. The accuracy of the integrated absorbed photon density for 1D TMM solutions in a 2D or 3D device can be controlled by various parameters (see [Accurate Absorbed Photon Density for 1D Optical Solvers on page 691](#)).

The `LayerStackExtraction` section determines how a 1D complex refractive index profile is extracted automatically from the grid file (see [Extracting the Layer Stack on page 686](#)). Based on this profile, the polarization-dependent optical intensity is calculated and interpolated onto the grid according to the referenced illumination window and the value of `PropagationDirection`. The optical generation profile then is calculated from the window-specific intensities. For overlapping windows, the intensity are summed in the region of intersection.

By default, the surrounding media at the top and bottom of the extracted layer stack are assumed to have the material properties of vacuum. However, the default can be overwritten by specifying a separate `Medium` section in the `LayerStackExtraction` section for the top and bottom of the layer stack if necessary. In the `Medium` section, a `Location` must be specified and a `Material`, or, alternatively, the values of `RefractiveIndex` and `ExtinctionCoefficient`:

```
LayerStackExtraction(
    Medium (
        Location = top
        Material = "Silicon"
```

## Chapter 21: Optical Generation

### Transfer Matrix Method

```
)  
Medium (  
    Location = bottom  
    RefractiveIndex = 1.4  
    ExtinctionCoefficient = 1e-3  
)  
)
```

If the optical generation profile is the sole quantity of interest, it is sufficient to specify the keyword `Optics` exclusively in the `Solve` section.

The TMM solver offers two options for computing the optical intensity from the complex field amplitudes of the forward-propagating and backward-propagating waves. Specifying `IntensityPattern=StandingWave` computes the optical intensity  $I$  as follows:

$$I = \text{Re}(A + B)^2 + \text{Im}(A + B)^2 \quad (762)$$

whereas using the syntax:

```
TMM (...  
    IntensityPattern = Envelope  
)
```

computes the optical intensity  $I$  using the following formula to prevent oscillations on the wavelength scale that might not be possible to resolve on the mixed-element simulation grid:

$$I = \text{Re}(A)^2 + \text{Im}(A)^2 + \text{Re}(B)^2 + \text{Im}(B)^2 \quad (763)$$

where  $A$  and  $B$  are the complex field amplitudes of the forward-propagating and backward-propagating waves, respectively. `IntensityPattern` can be specified globally, per region, or per material.

The keywords of `TMM` are summarized in [Table 268 on page 1654](#).

## Using Scattering Solver

To model the effect of scattering at rough interfaces, the scattering solver (see [Rough Surface Scattering on page 737](#)) must be configured in the command file, and the parameters characterizing each rough interface must be specified in the corresponding parameter file section.

### Command File Specification

The scattering solver is activated by specifying a `Scattering` section within the `TMM` section. By default, all interfaces are treated as rough. However, whether part of the light incident on a specific interface is actually scattered depends on the corresponding haze functions for reflection and transmission defined in the parameter file. A haze function evaluating to zero is equivalent to the interface being treated as flat.

## Chapter 21: Optical Generation

### Transfer Matrix Method

You can flag explicitly a specific region or material interface as not being rough as in the following example:

```
Physics {
    Optics (
        OpticalSolver (
            TMM (
                Scattering ( ... )
                RoughInterface #Default
            )
        )
    )
}
Physics (RegionInterface=<region1>/<region2>) {
    Optics (
        OpticalSolver (
            TMM ( -RoughInterface )
        )
    )
}
```

Sometimes, it might be more practical to set all interfaces as flat by specifying `-RoughInterface` in the global `Physics` section and only to label specific interfaces as rough in a corresponding region or material interface `Physics` section.

The angular discretization of the interval  $[-\pi/2, \pi/2]$  used for the ray bundles in the scattering solver can be set with the keyword `AngularDiscretization` in the `Scattering` section.

The keywords `Tolerance` and `MaxNumberOfIterations` determine the termination condition of the iterative scattering solver (see [Rough Surface Scattering on page 737](#)):

```
Physics {
    Optics (
        OpticalSolver (
            TMM (
                Scattering (
                    AngularDiscretization = 90
                    MaxNumberOfIterations = 150
                    Tolerance = 1e-3
                )
            )
        )
    )
}
```

For weakly absorbing structures, such as semiconductor layer stacks illuminated with light in the infrared part of the spectrum, many iterations might be necessary until the given `Tolerance` is reached. To detect such cases and to limit the total simulation time, it is advisable to adjust the value of `MaxNumberOfIterations`.

## Chapter 21: Optical Generation

### Transfer Matrix Method

#### Parameter File Specification

All parameters characterizing a rough interface are defined in the `OpticalSurfaceRoughness` section of the respective region or material interface section in the parameter file. The parameters can be classified into two groups: One relates to the specification of the haze functions and one contains the selection of the various angular distribution functions (ADFs).

For specifying the haze function for reflection (`HazeFunction_R`) and transmission (`HazeFunction_T`), two options are available:

- Constant: For each of the two haze functions  $H_R$  and  $H_T$ , a constant value can be set using the keywords `H_R` and `H_T`. This option can be useful if a simulation is performed at a single wavelength and the surface roughness is not varied as the haze functions usually depend on both.
- Analytic: The analytic expressions given by [Equation 759](#) and [Equation 760](#) are evaluated, where the surface roughness  $\sigma_{rms}$ , the exponents  $a_R$  and  $a_T$ , and the correction functions  $c_R$  and  $c_T$  are supplied by the user. In the simplest approximation,  $c_R$  and  $c_T$  are assumed to be constants over the whole wavelength range. However, in general, the correction functions are used to fit the analytic haze functions against experimental data. For that purpose, tabulated values are supported by setting `CorrectionFunction_R` or `CorrectionFunction_T` equal to `Table`.

Besides the haze functions, the ADFs are used to characterize the behavior of light incident at rough interfaces. The model supports the specification of different ADFs for direct incident light at a rough interface as well as for scattered light incident at a rough interface. Some ADFs depend on an additional parameter apart from the angle of incidence, whose value must be supplied by the user.

The detailed syntax including all options for the definition of the haze functions and ADFs are given in the following tables (where applicable, the default value is given at the beginning of the description).

Table 134 Specification of haze functions

Symbol	Keyword	Description
$H^r(\lambda, \varphi)$	<code>HazeFunction_R=Constant   Analytic</code>	Constant Type of haze function for reflection.
$H^t(\lambda, \varphi)$	<code>HazeFunction_T=Constant   Analytic</code>	Constant Type of haze function for transmission.
$H_R$	<code>H_R=&lt;float&gt;</code>	0 Constant value for haze function for reflection.
$H_T$	<code>H_T=&lt;float&gt;</code>	0 Constant value for haze function for transmission.

## Chapter 21: Optical Generation

### Transfer Matrix Method

*Table 134 Specification of haze functions (Continued)*

Symbol	Keyword	Description
$\sigma_{\text{rms}}$	Sigma=<float>	0 Optical surface roughness in [nm] used in analytic haze functions for reflection and transmission.
$a_R$	a_R=<float>	2 Exponent of analytic haze function for reflection.
$a_T$	a_T=<float>	3 Exponent of analytic haze function for transmission.
	CorrectionFunction_R=Constant   Table	Constant Type of correction function of analytic haze function for reflection.
	CorrectionFunction_T=Constant   Table	Constant Type of correction function of analytic haze function for transmission.
$c_R$	c_R=<float>	1 Constant value in interval [0 1] for correction function of analytic haze function for reflection.
$c_T$	c_T=<float>	1 Constant value in interval [0 1] for correction function of analytic haze function for transmission.
$c_R(\lambda)$	c_R(<table>)	Tabulated values for correction function of analytic haze function for reflection. First column contains wavelength in [ $\mu\text{m}$ ] and second column contains value of correction function in interval [0 1].
$c_T(\lambda)$	c_T(<table>)	Tabulated values for correction function of analytic haze function for transmission. First column contains wavelength in [ $\mu\text{m}$ ] and second column contains value of correction function in interval [0 1].
	TableInterpolation_c_R=Linear   Spline	Linear Type of interpolation used for tabulated values of correction function for analytic haze function for reflection.

## Chapter 21: Optical Generation

### Transfer Matrix Method

*Table 134 Specification of haze functions (Continued)*

Symbol	Keyword	Description
	TableInterpolation_c_T=Linear   Spline	Linear Type of interpolation used for tabulated values of correction function for analytic haze function for transmission.

*Table 135 Specification of angular distribution functions (not normalized)*

Symbol	Keyword	Description
$f_1(\phi)$	ADFDirect_R=Cosine   Constant   Ellipse   Gauss   Lorentz   Triangle	Cosine Type of angular distribution function for direct incident light reflected at a rough interface.
$C$	CDirect_R=<float>	2 Fitting parameter used in angular distribution function (where applicable) for direct incident light reflected at a rough interface.
$f_1(\phi)$	ADFDirect_T=Cosine   Constant   Ellipse   Gauss   Lorentz   Triangle	Cosine Type of angular distribution function for direct incident light transmitted through a rough interface.
$C$	CDirect_T=<float>	2 Fitting parameter used in angular distribution function (where applicable) for direct incident light transmitted through a rough interface.
$f_2(\phi)$	ADFSscatter_R=Constant   Cosine   Ellipse   Gauss   Lorentz   Triangle	Constant Type of angular distribution function for scattered light reflected at a rough interface.
$C$	CScatter_R=<float>	1 Fitting parameter used in angular distribution function (where applicable) for scattered light reflected at a rough interface.
$f_2(\phi)$	ADFSscatter_T=Constant   Cosine   Ellipse   Gauss   Lorentz   Triangle	Constant Type of angular distribution function for scattered light transmitted through a rough interface.

## Chapter 21: Optical Generation

### Transfer Matrix Method

Table 135 Specification of angular distribution functions (not normalized) (Continued)

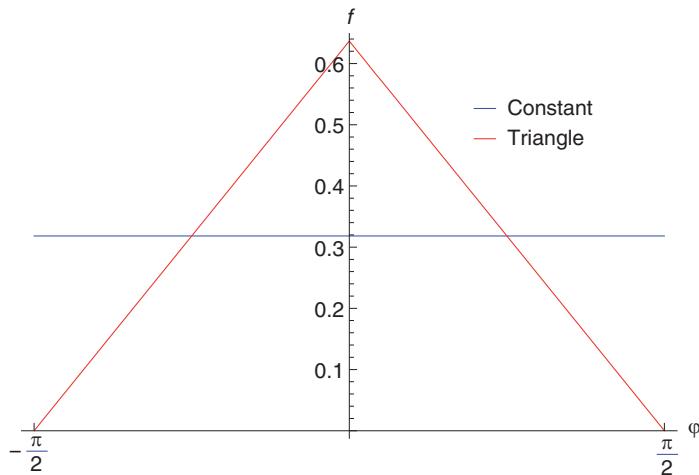
Symbol	Keyword	Description
$C$	<code>CScatter_T=&lt;float&gt;</code>	1 Fitting parameter used in angular distribution function (where applicable) for scattered light transmitted through a rough interface.

Table 136 Definition of ADF types used in Table 135

Type	Formula
Constant	$f(\phi) = 1$
Triangle	$f(\phi) = 1 - 2 \phi /\pi$
Gauss	$f(\phi) = e^{-\phi^2/(2C)}$
Lorentz	$f(\phi) = e^{1/(\phi^2 + C^2)}$
Cosine	$f(\phi) = \cos^C(\phi)$
Ellipse	$f(\phi) = \frac{\cos(\phi)}{\cos^2(\phi) + (2C)^{-2} \sin^2(\phi)}$

The following figures illustrate the various ADFs specified in Table 135.

Figure 41 ADF of type Constant and Triangle



## Chapter 21: Optical Generation

### Transfer Matrix Method

Figure 42 ADF of type Gauss

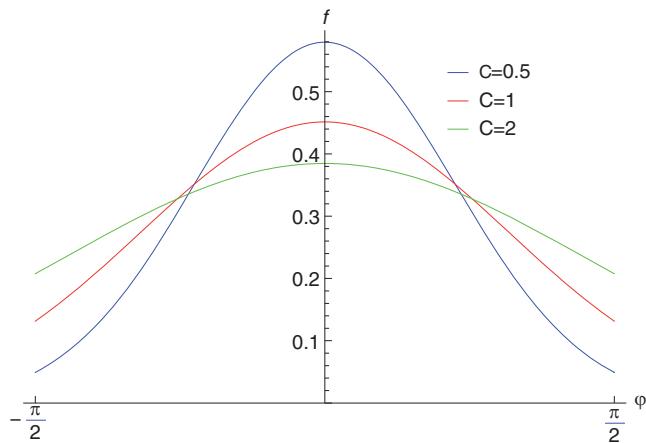


Figure 43 ADF of type Lorentz

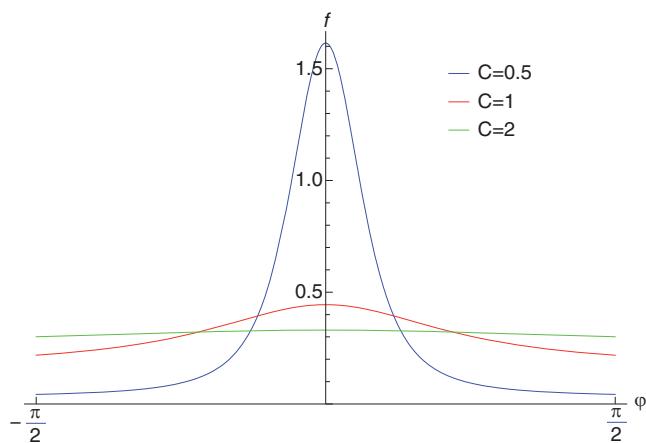
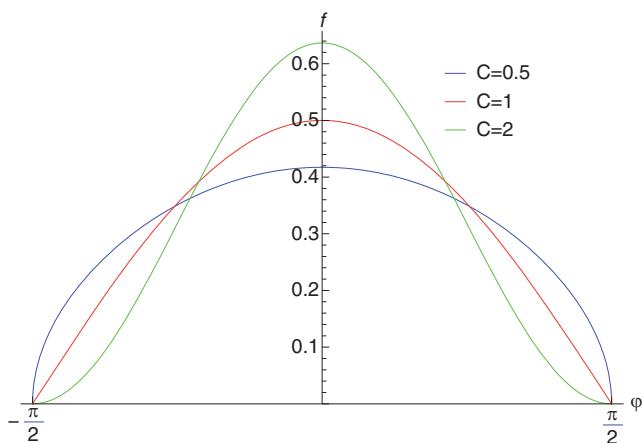


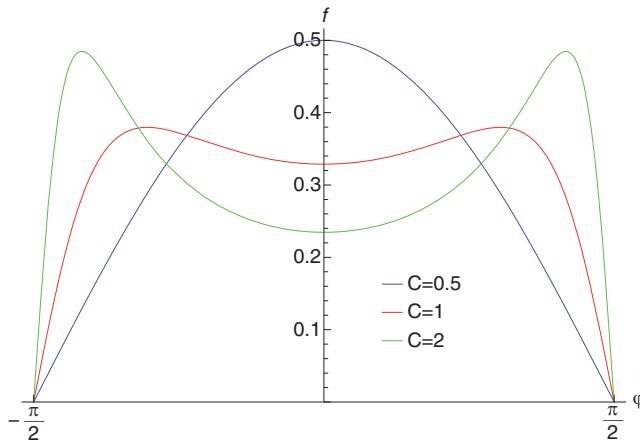
Figure 44 ADF of type Cosine



## Chapter 21: Optical Generation

### Transfer Matrix Method

Figure 45 ADF of type Ellipse



The following `OpticalSurfaceRoughness` section is an example for the specification of analytic haze functions and the default angular distribution functions:

```
OpticalSurfaceRoughness {
    HazeFunction_R=Analytic
    HazeFunction_T=Analytic
    Sigma=20    #[nm]
    a_R=2
    a_T=3
    CorrectionFunction_R=Table
    TableInterpolation_c_R=Spline
    c_R(
        0.3 0.04
        0.4 0.65
        0.8 0.9
        1.3 0.95
    )
    CorrectionFunction_T=Constant
    c_T=0.7

    ADFDirect_R=Cosine
    CDirect_R=2
    ADFDirect_T=Cosine
    CDirect_T=2
    ADFScatter_R=Constant
    ADFScatter_T=Constant
}
```

### Plot Quantities

When specifying `OpticalIntensity` and `AbsorbedPhotonDensity` in the `Plot` section of the command file, the following additional datasets are written to the plot file:

- `OpticalIntensityCoherent`: Specular part of optical intensity resulting from propagation of coherent light

## Chapter 21: Optical Generation

### Loading Solution of Optical Problem From Files

- `OpticalIntensityIncoherent`: Diffuse part of optical intensity resulting from propagation of incoherent light
- `AbsorbedPhotonDensityCoherent`: Specular part of absorbed photon density resulting from propagation of coherent light
- `AbsorbedPhotonDensityIncoherent`: Diffuse part of absorbed photon density resulting from propagation of incoherent light

---

## Loading Solution of Optical Problem From Files

In solar-cell and CIS simulations, it is often necessary to load several optical generation profiles as a function of one or more parameters, for example, excitation wavelength or angle of incidence, in Sentaurus Device. These profiles then are used to compute the response to each of them separately or to compute the response to the spectrum as a whole. A typical example is the computation of white-light generation using several generation profiles previously computed by Sentaurus Device Electromagnetic Wave Solver (EMW) or different wavelengths of the visible spectrum.

The parameters corresponding to a specific absorbed photon density or optical generation profile are written to the respective output file as special TDR tags as well. For 1D profiles generated with external tools, such tags can be specified in the file header (see [Importing 1D Profiles Into Higher-Dimensional Grids on page 752](#)).

This ensures that a profile can be identified later when loaded into a Sentaurus Device simulation. For example, if the optical problem is solved externally, it is not possible to determine the quantum yield, which is necessary to compute the optical generation based on the electronic properties of the underlying device structure. However, since the corresponding excitation wavelength is stored along with the optical solution, the quantum yield computation can be performed later in Sentaurus Device after loading the profile to compute the optical generation.

Absorbed photon density or optical generation profiles loaded into Sentaurus Device using the optical solver `FromFile` must meet one of the following requirements:

- Profiles must be saved in a TDR file.
- One-dimensional profiles that are to be imported into higher-dimensional grids must comply with the PLX file format described in [Importing 1D Profiles Into Higher-Dimensional Grids on page 752](#).

If the grid on which the profile is saved is not the same as that used for the device simulation (different mixed-element grid or arbitrary tensor grid arising from EMW simulation), the profile is interpolated automatically onto the simulation grid upon loading. For more details on how to control the interpolation, including the truncation and shifting of the interpolation

## Chapter 21: Optical Generation

### Loading Solution of Optical Problem From Files

domain, see [Controlling Interpolation When Loading Optical Generation Profiles on page 768](#).

The basic command file syntax for loading profiles from file is:

```
File {
    OpticalSolverInput = "<filename or filename pattern of profiles>"
}

Physics {
    Optics (
        OpticalGeneration (
            ComputeFromMonochromaticSource ()                      # or
            ComputeFromSpectrum ())
    )
    OpticalSolver (
        FromFile (
            DatasetName = AbsorbedPhotonDensity                 # Default
            SpectralInterpolation = Linear                      # Default Off
            IdentifyingParameter = ("Wavelength" "Theta")
        )
    )
}
}
```

#### Note:

The optical solver `FromFile` requires the specification of `ComputeFromMonochromaticSource` or `ComputeFromSpectrum` in the `OpticalGeneration` section. Otherwise, the computation of the optical generation based on the loaded profile is not activated.

The keyword `OpticalSolverInput` takes as its argument either the name of a single TDR or PLX file, or a UNIX-style file name pattern to select several such files whose names match the specified pattern. In both cases, a single file can contain more than one absorbed photon density or optical generation profile. In TDR files, the different profiles are saved in separate TDR states; whereas, in PLX files, they are simply listed sequentially, including their respective headers (see [Importing 1D Profiles Into Higher-Dimensional Grids on page 752](#)).

Each profile is characterized by its set of parameters, whose values must be unique among the profiles selected in the `File` section. As profiles can be characterized by different numbers of parameters, you must specify the ones to be considered for checking uniqueness by using the keyword `IdentifyingParameter`. Supported arguments are a simple parameter name such as `Wavelength` or, if ambiguous, a full parameter path such as `Optics/Excitation/Wavelength`.

You can also introduce new parameters, that is, parameters that do not exist in Sentaurus Device, by assigning them to a profile and referring to them in the list of `IdentifyingParameter`.

## Chapter 21: Optical Generation

### Loading Solution of Optical Problem From Files

Added parameters will appear under the following path:

```
Optics/OpticalSolver/FromFile/UserDefinedParameters/  
    <Name of added parameter>
```

User-defined parameters also are supported in the `IlluminationSpectrum` file in the `File` section (see [Illumination Spectrum on page 651](#)).

**Note:**

Only profiles that carry intensity as a parameter tag are supported for loading from files. Other profiles are ignored with a warning message. However, intensity must not be listed as `IdentifyingParameter` unless you want to enforce explicitly that all profiles are also unique with respect to their intensity.

The intensity associated with each profile and saved as a parameter tag in the respective file plays a special role as it is needed to scale the loaded profile according to the requested intensity. For `ComputeFromMonochromaticSource`, the loaded profile must be scaled to conform with the intensity specified in the `Excitation` section according to:

$$G = \frac{I_{\text{exc}}}{I_{\text{loaded}}} G_{\text{loaded}} \quad (764)$$

For `ComputeFromSpectrum`, the loaded profile for each spectral component is scaled to match the intensity given in the corresponding entry of the illumination spectrum file. The total optical generation is then computed as the sum over the scaled profiles given by:

$$G_{\text{tot}} = \sum_n \frac{I_{n, \text{spectrum}}}{I_{n, \text{loaded}}} G_{n, \text{loaded}} \quad (765)$$

When using the feature `ComputeFromSpectrum` (see [Illumination Spectrum on page 651](#)) or when ramping parameters, it is possible that, for a requested set of parameter values, no profile can be found that matches it. By default, the program will exit with an appropriate error message; however, you have two further options to deal with such situations. Using the keyword `SpectralInterpolation`, either `PiecewiseConstant` or `Linear` interpolation can be selected. Interpolation is based on the leading identifying parameter as opposed to complex multidimensional interpolation.

As mentioned earlier, two profile quantities are supported, absorbed photon density and optical generation, which can be selected using the keyword `DatasetName`. The choice determines whether a quantum yield model (see [Quantum Yield Models on page 658](#)) will be applied. If optical generation is specified, quantum yield is assumed to be 1. Otherwise, the corresponding quantum yield model specified in the `OpticalGeneration` section is used to compute the optical generation profile from the loaded absorbed photon density.

## Importing 1D Profiles Into Higher-Dimensional Grids

Importing 1D profiles into higher-dimensional grids is supported for files in PLX format. The following example documents the syntax of PLX files for two 1D profiles characterized by three parameters in a single file:

```
# some comments
"<optional string for tagging a profile>"
Wavelength = 0.5 [um] Theta = 0 [deg] Intensity = 0.1 [W*cm^-2]
0.0 1e19
0.8 1e20
1.4 1e17
2.0 5e18

# some comments
"<optional string for tagging a profile>"
Wavelength = 0.6 [um] Theta = 30[deg] Intensity = 0.1 [W*cm^-2]
0.0 1e19
0.8 1e20
1.4 1e17
2.0 5e18
...
```

Each profile can be preceded by an optional comment starting with the hash character (#), an optional tag given in double quotation marks, and a list of parameters with their corresponding value and unit as shown in the example above. If a tag is supplied, then it will be used as a curve name when visualizing the file in Inspect, which also supports PLX files containing several profiles.

After the 1D profiles have been loaded into Sentaurus Device, they must be interpolated onto the 2D or 3D grid. Similar to the TMM solver (see [Using Transfer Matrix Method on page 739](#)), the concept of an illumination window is applied for this purpose and it requires the specification of at least one `Window` section in the `Excitation` section. See [Illumination Window on page 676](#) for further details. The accuracy of the integrated APD for 1D profiles in a 2D or 3D device can be controlled by various parameters (see [Accurate Absorbed Photon Density for 1D Optical Solvers on page 691](#)).

**Note:**

For the interpolation of the 1D profile onto the 2D or 3D grid, the first data point of a profile is always pinned to the illumination window plane, independent of its actual coordinate value.

---

## Ramping Profile Index

The optical solver `FromFile` supports a keyword `ProfileIndex`, which can be used to address a certain profile directly by its index instead of its identifying parameters. The index

## Chapter 21: Optical Generation

### Optical Beam Absorption Method

is derived from all valid profiles sorted according to the value of the leading identifying parameter specified in the command file.

This feature can be used to ramp through all available profiles in a Quasistationary statement without having to know the exact parameter values for each profile. The spectral interpolation options described in [Loading Solution of Optical Problem From Files on page 749](#) also are supported for ramping the ProfileIndex. If an index value is requested in a Quasistationary statement that exceeds the number of valid profiles, then the Quasistationary is finished and control is given to the following statement in the Solve section.

The following example shows the ramping of the ProfileIndex:

```
File {
    OpticalSolverInput = "absorbed_photon_density_input_*file.tdr"
}
Physics {
    OpticalGeneration (
        ComputeFromMonochromaticSource ( )
    )
    OpticalSolver (
        FromFile (
            ProfileIndex = 0
            DatasetName = AbsorbedPhotonDensity
            SpectralInterpolation = Off
            IdentifyingParameter = ("Optics/Excitation/Wavelength"
                "Theta" "Phi")
        )
    )
}
Solve {
    Quasistationary (
        InitialStep=0.01 MaxStep=0.01 MinStep=0.01
        Plot { Range = ( 0., 1 ) Intervals = 5 }
        Goal { ModelParameter="ProfileIndex" value=100 }
    ) { Optics }
}
```

Note that the counting of profiles starts with index zero. For more information about ramping parameters, see [Parameter Ramping on page 689](#).

---

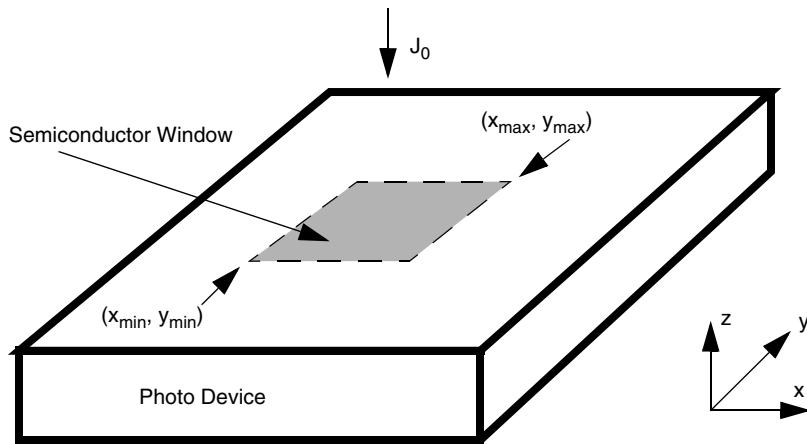
## Optical Beam Absorption Method

The optical beam absorption method in Sentaurus Device computes optical generation by simple photon absorption using Beer's law. Thereby, multiple optical beams can be defined to represent the incident light.

## Physical Model

Figure 46 illustrates one optical beam and its optical intensity distribution in a 3D device.  $J_0$  denotes the optical beam intensity (number of photons that cross an area of  $1 \text{ cm}^2$  per  $1 \text{ s}$ ) incident on the semiconductor device.  $(x_{\min}, y_{\min})$  and  $(x_{\max}, y_{\max})$  define the rectangular illumination window. The space shape  $F_{xy}$  of the incident beam intensity is defined by the illumination window, where  $F_{xy}$  is equal to 1 inside of it and zero otherwise.

*Figure 46 Intensity distribution of an optical beam modeled by a rectangular illumination window*



The following equations describe useful relations for the photogeneration problem:

$$E_{\text{ph}} = \frac{hc}{\lambda} \quad (766)$$

$$J_0 = \frac{P_0}{E_{\text{ph}}}$$

where:

- $P_0$  is the incident wave power per area [ $\text{W/cm}^2$ ].
- $\lambda$  is the wavelength [cm].
- $h$  is Planck's constant [Js].
- $c$  is the speed of light in a vacuum [cm/s].
- $E_{\text{ph}}$  is the photon energy that is approximately equal to  $\frac{1.24}{\lambda[\mu\text{m}]}$  in eV.

## Chapter 21: Optical Generation

### Optical Beam Absorption Method

The optical beam absorption model computes the optical generation rate along the z-axis taking into account that the absorption coefficient varies along the propagation direction of the beam according to:

$$G^{\text{opt}}(z, t) = J_0 F_t(t) F_{xy} \cdot \alpha(\lambda, z) \cdot \exp \left[ - \int_{z_0}^z \alpha(\lambda, z') dz' \right] \quad (767)$$

where:

- $t$  is the time.
- $F_t(t)$  is the beam time behavior function. For a Gaussian pulse, it is equal to 1 for  $t$  in  $[t_{\min}, t_{\max}]$  and shows a Gaussian distribution decay outside the interval with the standard deviation  $\sigma_t$ .
- $z_0$  is the coordinate of the semiconductor surface.
- $\alpha(\lambda, z)$  is the nonuniform absorption coefficient along the z-axis.

As described in the following section, the optical beam absorption method supports a wide range of window shapes (see [Illumination Window on page 676](#)) as well as arbitrary beam time behavior functions (see [Specifying Time Dependency for Transient Simulations on page 661](#)). It is also not limited to beams propagating along the z-axis. The example above has only been used for demonstration purposes.

---

## Using Optical Beam Absorption Method

The optical beam absorption method is activated by the optical solver `OptBeam`. The profile of the absorption coefficient used in [Equation 767](#) is determined by the `LayerStackExtraction` section inside the `OptBeam` section; whereas, the shape of the beam is specified by the illumination window. The wavelength and intensity of the incident light are defined in the `Excitation` section.

The required syntax of the optical beam absorption method is summarized here:

```
Physics {
    ...
    Optics (
        ...
        Excitation (
            Wavelength = 0.5
            Intensity = 0.1
            Window("L1") (
                <>window options>
            )
        )
        OpticalSolver (
            OptBeam (
                LayerStackExtraction (
                    WindowName = "L1"

```

## Chapter 21: Optical Generation

### Beam Propagation Method

```
        <layer stack extraction options>
    )
)
)
}
}
```

Further details about the `Window` and `LayerStackExtraction` sections can be found in [Illumination Window on page 676](#) and [Extracting the Layer Stack on page 686](#), respectively.

The accuracy of the integrated absorbed photon density for 1D profiles in a 2D or 3D device can be controlled by various parameters (see [Accurate Absorbed Photon Density for 1D Optical Solvers on page 691](#)).

#### Note:

You can simulate several beams at the same time by specifying the corresponding number of `Window` and `LayerStackExtraction` sections. However, all beams are assigned the same wavelength and intensity, which is specified in the common `Excitation` section.

---

## Beam Propagation Method

In Sentaurus Device, the beam propagation method (BPM) can be applied to find the light propagation and penetration into devices such as photodetectors. Despite being an approximate method, its efficiency and relative accuracy make it attractive for bridging the gap between the raytracer and the FDTD solver (EMW) featured in Sentaurus Device. The BPM solver is available for both 2D and 3D device geometries, where its computational efficiency compared with a full-wave approach becomes particularly apparent in three dimensions.

#### Note:

Do not use BPM for 3D CMOS image sensors because BPM cannot resolve the polarization transformation caused by the 3D lens.

---

## Physical Model

The BPM implemented in Sentaurus Device is based on the fast Fourier transform (FFT) and is a variant of the FFT BPM, which was developed by Feit and Fleck [7].

The solution of the scalar Helmholtz equation:

$$\left[ \nabla_t^2 + \frac{\partial^2}{\partial z^2} + k_0^2 n^2(x, y, z) \right] \phi(x, y, z) = 0 \quad (768)$$

## Chapter 21: Optical Generation

### Beam Propagation Method

at  $z + \Delta z$  with  $\nabla_t^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$  and  $n(x, y, z)$  being the complex refractive index can be written as:

$$\frac{\partial}{\partial z} \phi(x, y, z + \Delta z) = \mp i \sqrt{k_0^2 n^2 + \nabla_t^2} \phi(x, y, z) = \pm i \wp \phi(x, y, z) \quad (769)$$

In the paraxial approximation, the operator  $\wp$  reduces to:

$$\wp \equiv \sqrt{k_0^2 n_0^2 + \nabla_t^2} + k_0 \delta n \quad (770)$$

where  $n_0$  is taken as a constant reference refractive index in every transverse plane. By expressing the field  $\phi$  at  $z$  as a spatial Fourier decomposition of plane waves, the solution to [Equation 768](#) for forward-propagating waves reads:

$$\phi(x, y, z + \Delta z) = \frac{1}{(2\pi)^2} \exp(i k_0 \delta n \Delta z) \sum_{-\infty}^{\infty} d\vec{k}_t \exp(i \vec{k}_t \cdot \vec{r}) \exp(i \sqrt{k_0^2 n_0^2 - \vec{k}_t^2} \Delta z) \tilde{\phi}(\vec{k}_t, z) \quad (771)$$

where  $\tilde{\phi}$  denotes the transverse spatial Fourier transform. As can be seen from [Equation 771](#), each Fourier component experiences a phase shift, which represents the propagation in a medium characterized by the reference refractive index  $n_0$ . The phase-shifted Fourier wave is then inverse-transformed and given an additional phase shift to account for the refractive index inhomogeneity at each  $(x, y, z + \Delta z)$  position. In the numeric implementation of [Equation 771](#), an FFT algorithm is used to compute the forward and inverse Fourier transform.

## Bidirectional BPM

The bidirectional algorithm is based on two operators as described by KaczmarSKI and Lagasse [\[8\]](#). The first operator defines a unidirectional propagation, which is outlined in the previous section. The second operator accounts for the reflections at interfaces of the refractive index along the propagation direction. In a first pass, the reflections at all interfaces are computed. The following pass in the opposite direction adds these contributions to the propagating field and calculates the reflections of the forward-propagating field. By iterating this procedure until convergence is reached, a self-consistent algorithm is established.

## Boundary Conditions

Due to the finite size of the computational domain, appropriate boundary conditions must be chosen, which minimize any numeric errors in the propagation of the optical field related to boundary effects. In the standard FFT BPM, any waves propagating through a boundary will reappear as a new perturbation at the opposite side of the computation window, effectively representing periodic boundary conditions. In situations where the optical field vanishes at the domain boundaries, this effect can be neglected. In other cases, an absorbing boundary condition is needed.

## Chapter 21: Optical Generation

### Beam Propagation Method

Perfectly matched layers (PMLs) boundary conditions have the advantage of absorbing the optical field when it reaches the domain boundaries. The BPM implemented in Sentaurus Device supports the PML boundary condition, which has been developed for continuum spectra.

The formulation is based on the introduction of a stretching operator:

$$\nabla_s \equiv \frac{1}{S_x} \hat{x} \frac{\partial}{\partial x} + \frac{1}{S_y} \hat{y} \frac{\partial}{\partial y} + \frac{1}{S_z} \hat{z} \frac{\partial}{\partial z} \quad (772)$$

from which an equivalent set of the Maxwell equations can be derived. In [Equation 772](#),  $S_x$ ,  $S_y$ , and  $S_z$  are called stretching parameters, which are complex numbers defined in different PML regions. In non-PML regions, these stretching parameters are equal to one. The modified propagation equation then reads:

$$\phi(x, y, z + \Delta z) = \frac{1}{(2\pi)^2} \exp(iS_z k_0 \delta n \Delta z) \sum_{-\infty}^{\infty} d\vec{k}_t \exp(i\vec{k}_t \cdot \vec{r}) \exp(iS_z) \exp(i\sqrt{k_0^2 n_0^2 - \vec{k}_t^2} \Delta z) \tilde{\phi}(\vec{k}_t, z) \quad (773)$$

where  $\hat{k}_t^2 = (k_x/S_x)^2 + (k_y/S_y)^2$  is the square of the stretched transverse wave number.

The stretching parameters can be fine-tuned to minimize spurious reflections.

---

## Using Beam Propagation Method

This section discusses how to use the beam propagation method.

### General

The following code excerpt describes the general setup for using the scalar BPM solver to compute the optical generation. The computation of the optical generation is activated by an `OpticalGeneration` section, in which the `QuantumYield` model, as defined in [Equation 753](#) and [Equation 754](#), can be specified. Solver-specific parameters are listed in the `BPM` section. The excitation parameters are split into the general `Excitation` section for parameters common to all optical solvers and the `BPM Excitation` section. In the latter section, you can select either a `PlaneWave` or `Gaussian` excitation.

The `GridNodes` parameter determines the number of discretization points for each spatial dimension. In the next line, the reference refractive index is specified that is used in the operator expansion in [Equation 770](#).

## Chapter 21: Optical Generation

### Beam Propagation Method

The parameters related to the excitation field and the boundary conditions for each spatial dimension are grouped in subsections as explained here:

```
Physics {
    ...
    Optics (
        OpticalGeneration (
            QuantumYield (
                ...
            )
            OpticalSolver (
                BPM (
                    GridNodes = (256,256,1600)
                    ReferenceRefractiveIndex = 2.2
                    Excitation (
                        ...
                    )
                    Boundary ( ... )
                )
            )
            Excitation (
                Wavelength = 0.5
                Intensity = 0.1
                Theta = 0.0
            )
        )
    )
}
```

As the reference refractive index can greatly influence the accuracy of the solution due its use in the expansion of the propagation operator, several options are available for its specification. The simplest option is to specify a globally constant value as shown above. For multilayer structures with large refractive index contrasts, better results can be achieved by using either the average or the maximum value of the refractive index in each propagation plane. In some cases, a reference refractive index that is computed as the field-weighted average of the refractive index in each propagation plane might be the best option. In addition to these options, a global offset can be specified to further optimize the results. For details about selecting the various options, see [Table 240 on page 1632](#).

## Bidirectional BPM

The bidirectional BPM solver is activated by specifying a `Bidirectional` section in the `BPM` section:

```
Optics (
    OpticalSolver (
        BPM (
            ...
            Bidirectional (
                Iterations = 3
                Error = 1e-3
            )
        )
    )
)
```

## Chapter 21: Optical Generation

### Beam Propagation Method

```
)  
)  
)
```

In the Bidirectional section, two break criteria can be set to control the total number of forward and backward passes that are performed in the beam propagation method. If the number of iterations exceeds the value of `Iterations` or if the relative error with respect to the previous iteration is smaller than the value of `Error`, it is assumed that the solution has been found. Note that a single iteration can be either a forward or backward pass.

From a performance point of view, it is important to mention that consecutive iterations only require a fraction of CPU time compared with the initial pass.

For plotting the optical field after every iteration, the keyword `OpticalField` must be listed in the `Plot` section of the command file. If only the final optical intensity is of interest, it is sufficient to specify the keyword `OpticalIntensity` in the `Plot` section.

## Excitation

In the beam propagation method, a given input field, which is referred to as excitation in the remainder of this section, is propagated through the device structure. Two types of excitation are supported: a Gaussian and a (truncated) PlaneWave. Both are characterized by a propagation direction, wavelength, and power as shown below. In two dimensions, the propagation direction is determined by the angle between the positive y-axis and the propagation vector. To specify the propagation direction in three dimensions, two angles, `Theta` and `Phi`, must be given. Their definition is illustrated in [Figure 47](#).

### Note:

The propagation direction must be along the z-direction (nonzero z-component of the propagation vector) in three dimensions and along the y-direction (nonzero y-component of the propagation vector) in two dimensions.

The incident light power in the plane wave is specified as the `Intensity` in  $[W/cm^2]$ . For a Gaussian excitation, the given value refers to its maximum. All of these parameters are listed in the global `Excitation` section.

The excitation parameters common to all types of excitation are:

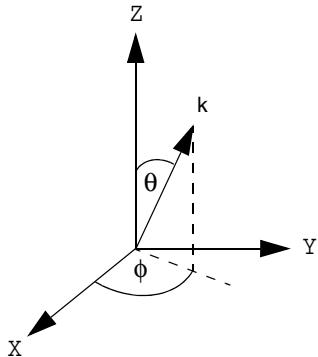
```
Physics {  
    ...  
    Optics (  
        Excitation (  
            Theta = 10.0          # [degree]  
            Phi = 60.0           # [degree], only in 3D  
            Wavelength = 0.3     # [um]  
            Intensity = 0.1       # [W/cm^2]  
            ...  
        )  
        ...  
    )
```

## Chapter 21: Optical Generation

### Beam Propagation Method

```
)  
}
```

Figure 47 Definition of angles for specification of propagation direction in three dimensions



A Gaussian excitation along the propagation direction defined by  $k$  is determined by its half-width SigmaGauss [ $\mu\text{m}$ ] and its center position CenterGauss [ $\mu\text{m}$ ] as shown below for a 2D and 3D device geometry.

Gaussian excitation (2D):

```
Physics {  
    ...  
    Optics(  
        OpticalSolver (  
            BPM (  
                Excitation (  
                    Type = "Gaussian"  
                    SigmaGauss = (2.0)      # [ $\mu\text{m}$ ]  
                    CenterGauss = (0.0)    # [ $\mu\text{m}$ ]  
                )  
            )  
        )  
    ...  
}
```

Gaussian excitation (3D):

```
Physics {  
    ...  
    Optics(  
        OpticalSolver (  
            BPM (  
                Excitation (  
                    Type = "Gaussian"  
                    SigmaGauss = (2.0,4.0)    # [ $\mu\text{m}$ ]  
                    CenterGauss = (0.0,1.0)    # [ $\mu\text{m}$ ]  
                )  
            )  
        )  
    ...  
}
```

## Chapter 21: Optical Generation

### Beam Propagation Method

```

        )
    )
    ...
}
}
```

For a plane wave excitation, it is possible to specify the truncation positions for each coordinate axis as well as a parameter that determines the fall-off. For numeric reasons, a discrete truncation must be avoided. Instead, a Fermi function-like fall-off is available:

$$F_{xy} = \left[ 1 + \exp \frac{|\tilde{x}| - x_0}{s_x} \right]^{-1} \cdot \left[ 1 + \exp \frac{|\tilde{y}| - y_0}{s_y} \right]^{-1} \quad (774)$$

$$\phi(x, y, z=0) = F_{xy} \cdot \phi_0$$

which can be tuned by adjusting the TruncationSlope and TruncationPosition parameters in the Excitation section. In [Equation 774](#),  $\phi_0$  is the amplitude of the incident light,  $s_{x,y}$  denotes the TruncationSlope,  $x, y$  is the position with respect to the symmetry point, and  $x_0, y_0$  is the half width. Both  $x, y$  and  $x_0, y_0$  are deduced from the TruncationPosition parameters. In two dimensions, the second factor on the right-hand side in [Equation 774](#) is omitted.

Truncated plane wave excitation (2D):

```

Physics {
    Optics (
        OpticalSolver (
            BPM (
                Excitation (
                    Type = "PlaneWave"
                    TruncationPositionX = (-1.0,1.0)
                    TruncationSlope = (0.3)
                )
            )
        )
    ...
}
}
```

Truncated plane wave excitation (3D):

```

Physics {
    Optics (
        OpticalSolver (
            BPM (
                Excitation (
                    Type = "PlaneWave"
                    TruncationPositionX = (-1.0,1.0)
                    TruncationPositionY = (-2.0,3.0)
                    TruncationSlope = (0.3,0.3)
                )
            )
        )
}
```

## Chapter 21: Optical Generation

### Beam Propagation Method

```
        )
    )
}
```

## Boundary

For every spatial dimension, a separate boundary condition and corresponding parameters can be defined. Periodic boundary conditions inherent to the FFT BPM solver are the default in the transverse dimensions. The specification of PML boundary conditions in the x-direction and y-direction is shown below. With the keyword `Order`, the spatial variation of the complex stretching parameter can be specified. In this example, a quadratic increase from the minimum value to the maximum value within the given number of `GridNodes` on either side has been chosen.

Optionally, several `VacuumGridNodes` can be inserted between the physical simulation domain and the PML boundary:

```
Physics {
    Optics(
        OpticalSolver (
            BPM (
                Boundary (
                    Type = "PML"
                    Side = "X"
                    Order = 2
                    StretchingParameterReal = (1.0,1,0)
                    StretchingParameterImag = (1.0,5.0)
                    GridNodes = (5,5)
                    VacuumGridNodes = (4,4)
                )
                Boundary (
                    Type = "PML"
                    Side = "Y"
                    Order = 2
                    StretchingParameterReal = (1.0,1,0)
                    StretchingParameterImag = (1.0,4.0)
                    GridNodes = (8,8)
                )
            )
        )
    ...
}
```

## Ramping Input Parameters

Several input parameters of the BPM solver such as the wavelength of the incident light can be ramped to obtain device characteristics as a function of the specified parameter. The general concept of ramping the values of physical parameters is described in [Ramping Physical Parameter Values on page 126](#).

## Chapter 21: Optical Generation

### Beam Propagation Method

For details about ramping `Optics` parameters, see [Parameter Ramping on page 689](#).

## Visualizing Results on Native Tensor Grid

For an accurate analysis of the BPM results, the complex refractive index, the complex optical field, and the optical intensity can be plotted on the native tensor grid. In general, the results from the BPM solver are interpolated from a tensor grid to a mixed-element grid on which the device simulation is performed.

For physical and numeric reasons, the mesh resolution of the optical grid needs to be much finer than that of the electrical grid in most regions of the device. This can lead to the introduction of interpolation errors, for example, by undersampling the respective dataset on the electrical grid. To obtain an estimate of the interpolation error, it can be beneficial to visualize the BPM results on its native tensor grid. As typical tensor grids in three dimensions tend to be very large, it is also possible to extract a limited domain for more efficient visualization. In two dimensions, a rectangular subregion or an axis-aligned straight line can be plotted; whereas in three dimensions, a subvolume, a plane perpendicular to the coordinate axes, and a straight line can be visualized directly on the tensor grid.

Tensor plots can be activated by specifying one or more `TensorPlot` sections and an optional base name for the tensor-plot file name in the `File` section.

The following syntax extracts a plane perpendicular to the z-axis at position  $Z = 1.3 \mu\text{m}$ , which extends from  $-2 \mu\text{m}$  to  $3 \mu\text{m}$ , and from  $-1 \mu\text{m}$  to  $5 \mu\text{m}$  in the x- and y-direction, respectively:

```
File (
    ...
    TensorPlot = "tensor_plot"
)
...
TensorPlot (
    Name = "plane_z_const"
    Zconst = 1.3
    Xmin = -2
    Xmax = 3
    Ymin = -1
    Ymax = 5
) {
    ComplexRefractiveIndex
    OpticalField
    OpticalIntensity
}
```

To extract a box in three dimensions that extends over the whole device horizontally but is bound in the vertical direction, the `TensorPlot` section looks like:

```
TensorPlot (
    Name = "bounded_box"
    Zmin = 3
```

## Chapter 21: Optical Generation

### Composite Method

```
    Zmax = 8
)
{
    ...
}
```

---

## Composite Method

The composite method is an optical solver that composes an optical solution by summing the results of other optical solvers. Its flexibility targets applications with more complex optical setups that can be modeled as a linear superposition of individual solution components.

The unified interface for optical generation computation is limited in that the contribution from `ComputeFromMonochromaticSource` and `ComputeFromSpectrum` to the total optical generation rate is computed by the same optical solver. While some optical solvers allow the definition of different illumination windows, they still must share the same excitation parameters. It is also not possible to specify several `ComputeFromMonochromaticSource` and `ComputeFromSpectrum` sections at the same time to model, for example, different light sources illuminating a device. Sometimes you can work around these limitations by precomputing the illumination by one light source and loading it in a subsequent simulation using `ReadFromFile`. However, it typically results in more complex simulation setups and tool flows.

With the composite method, these limitations can be overcome. Its implementation is based on the mixed-mode simulation framework that requires the creation of a separate optical device instance for each optical solver. An optical device instance is essentially a device instance limited to the computation of an optical problem. For details about mixed-mode in Sentaurus Device, see [Chapter 3 on page 90](#).

---

## Using the Composite Method

A typical simulation setup requires the definition of a named master `Device` or `OpticalDevice` (in the case of optics standalone simulations), and one or more additional definitions of `OpticalDevice`.

### Note:

Syntactically, there is no difference between a named `Device` and a named `OpticalDevice` section at the top level in the command file. However, instances created from these definitions fulfill different tasks.

In the master, the composite method is specified as the optical solver in the `Optics` section:

```
Device diode {
    File { ... }
    Electrode { ... }
```

## Chapter 21: Optical Generation

### Composite Method

```
Plot { ... }
Math { ... }
Physics {
    ...
    Optics (
        ...
        OpticalSolver (Composite)
    )
}
OpticalDevice optdevice {
    File { ... }
    Plot { ... }
    Math { ... }
    Physics {
        Optics (
            Excitation (
                Wavelength = 0.3
                Theta = 30
                Intensity = 0.1
            )
            OpticalSolver ( TMM (...) )
            ComplexRefractiveIndex ( ... )
            OpticalGeneration (
                ...
                ComputeFromMonochromaticSource ( ... )
                ComputeFromSpectrum ( ... )
            )
        )
    }
}
```

Any of the other optical solvers listed in [Specifying the Optical Solver on page 669](#) can be specified in the `OpticalDevice` definitions whose instances are accessed by the composite method. You can create more than one instance from a given `OpticalDevice` definition and overwrite specific parameters in the instantiation statement.

This is particularly useful for modeling different illumination sources that share many common properties. For example:

```
System {
    diode d1 (
        "p-contact" = vdd "n-contact" = gnd
        OpticalDevice = [ "tmm1" "tmm2" "tmm3" ]
    ) { ... }
    optdevice tmm1 () {Physics {Optics (Excitation (Wavelength=0.4) ) } }
    optdevice tmm2 () {Physics {Optics (Excitation (Theta=50) ) } }
    optdevice tmm3 () {Physics {Optics (Excitation (Intensity=0.2) ) } }
}
```

In this example, the device instance `d1` is associated with the optical device instances `tmm1`, `tmm2`, and `tmm3` by the `OpticalDevice` statement. These are the optical device instances

## Chapter 21: Optical Generation

### Composite Method

that the `Composite solver` defined in `Device diode` will consider when composing its solution.

In the `Solve` section, the different instances can be addressed separately by preceding the equation name with the instance name followed by a dot, or as a whole by specifying the equation only as shown in the optics standalone example:

```
Solve {  
    tmm1.Optics  
    Coupled (d1.Optics tmm2.Optics tmm3.Optics)  
    Optics  
}
```

In optoelectronic simulations, the `Optics` equation can be omitted and is solved implicitly (considering all device and optical device instances) if the corresponding optical generation models are activated:

```
Solve {  
    Quasistationary (  
        InitialStep = 1 MaxStep = 1 MinStep = 1  
        Goal { OpticalDevice= "tmm1" ModelParameter= "Wavelength" = 0.8 }  
        Goal { OpticalDevice= "tmm3" ModelParameter= "Wavelength" = 0.4 }  
    ) {  
        Coupled { Poisson Electron Hole }  
    }  
}
```

#### Note:

The `Coupled` statement can contain either electrical transport equations such as `Poisson`, `Electron`, and `Hole`, or several instance-specific `Optics` equations, but not a combination of both. If you want to explicitly call `Optics` and electrical transport equations, you must use a `Plugin` statement.

Generally, simulation results such as spatial-dependent plots and current plots are produced for each `Device` or `OpticalDevice` instance according to the respective `Plot` and `CurrentPlot` sections. However, for ease of use, the most relevant spatial-dependent datasets from `OpticalDevice` instances used by the `Composite solver` are written automatically to corresponding TDR files.

Such datasets have their original dataset name appended with `FromInstance_<instance_name>`, for example, `OpticalGenerationFromInstance_tmm1`.

To access `CurrentPlot` quantities from one `Device` or `OpticalDevice` in another `Device` or `OpticalDevice`, use either the `Device="<name>"` or `OpticalDevice="<name>"` specification in the `CurrentPlot` section.

## Controlling Interpolation When Loading Optical Generation Profiles

You can load optical generation and absorbed photon density profiles into Sentaurus Device using either of the following features:

- [Loading and Saving Optical Generation From and to Files](#)
- [Loading Solution of Optical Problem From Files](#)

If the optical generation or absorbed photon density profile to be loaded is defined on a grid that is different from the one used in the device simulation, it is interpolated automatically onto the simulation grid upon loading. The source grid can be either a mixed-element grid or a tensor grid resulting from an EMW simulation. By default, vertex-based linear interpolation is used.

For mixed-element to mixed-element interpolation, a conservative interpolation algorithm is supported [9], which guarantees that the integral of the interpolation quantity on the source grid is the same as on the destination grid. To select this option, set `GridInterpolation` to `Conservative` instead of `Simple`.

The result of the interpolation can be further controlled by specifying a relative shift between the source and the destination grid, as well as by restricting the source and destination domain used for the interpolation. This feature is useful when the optical solution has been computed on a smaller or larger domain than the electrical simulation grid and, therefore, must be aligned accordingly.

The `ShiftVector` is specified directly in the `ReadFromFile` or `FromFile` section; whereas, the source and destination domain is specified in a section called `ImportDomain`.

A domain can be defined as either a list of regions or a box given by its lower-left and upper-right corners as shown in this example:

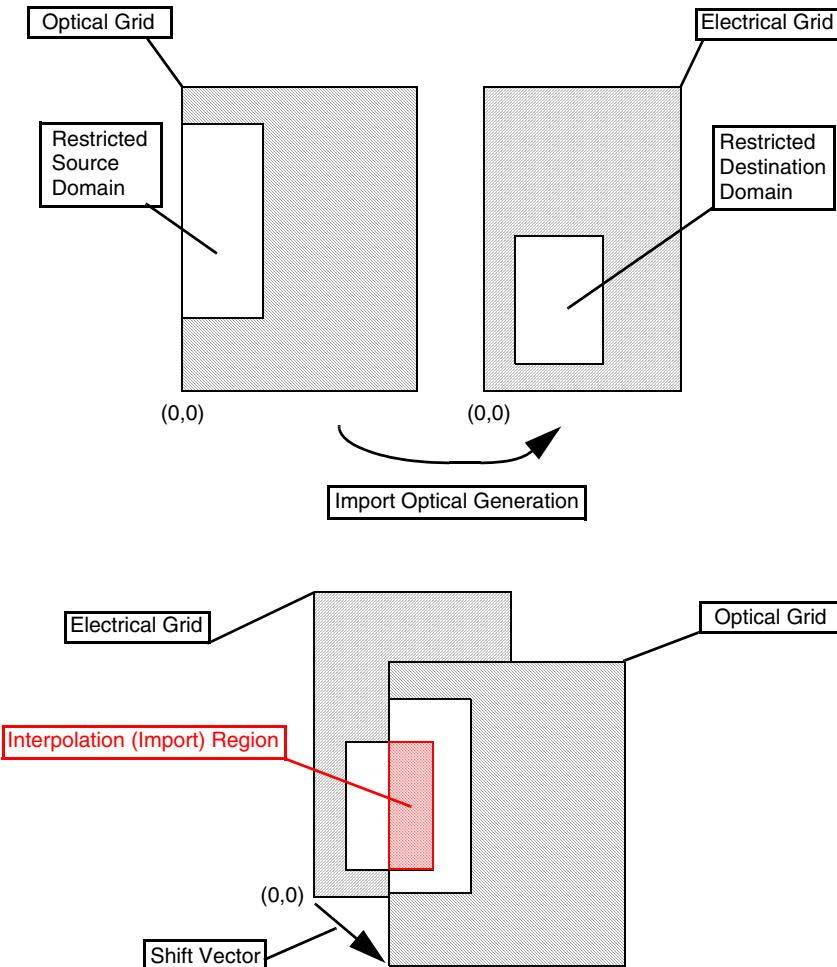
```
Physics {
    Optics (
        OpticalGeneration (
            ReadFromFile (
                GridInterpolation = Conservative
                ShiftVector = (0.5 1 2)
                ImportDomain (
                    SourceRegions = ("substrate", "region_1")
                    DestinationBoxCorner1 = (0.5 0.5 1)
                    DestinationBoxCorner2 = (2 2 4)
                )
            )
        )
    )
}
```

## Chapter 21: Optical Generation

### Controlling Interpolation When Loading Optical Generation Profiles

**Figure 48** illustrates the concept of domain truncation using source and destination domains, as well as applying a relative shift between the optical and electrical grids.

**Figure 48** Illustration of domain truncation and shifting of source and destination grids used in interpolation



The interpolation algorithm always interpolates the values even if the material of the source and destination grid is different at a certain vertex. However, optical generation is always set to zero in nonsemiconductor regions irrespective of the source profile. For details about specifying the source and destination domain in the `ImportDomain` section, see [Table 247 on page 1637](#).

**Note:**

`GridInterpolation`, `ShiftVector`, and `ImportDomain` are not supported for the importing of 1D profiles because the [Illumination Window on page 676](#) already provides similar functionality.

## Chapter 21: Optical Generation

### Optical AC Analysis

Region domains specified in the `ImportDomain` section are supported only for source grids that are mixed-element type or tensor type, containing region information (the `Extractor` section in EMW must contain `Region` in the `Quantity` list).

---

## Optical AC Analysis

An optical AC analysis calculates the quantum efficiency as a function of the frequency of the optical signal intensity. The method is based on the AC analysis technique and provides real and imaginary parts of the quantum efficiency versus the frequency.

During an optical AC analysis, a small perturbation of the incident wave power  $\delta P_0$  is applied. Therefore, the photogeneration rate is perturbated as  $G^{\text{opt}} + \delta G^{\text{opt}} e^{i\omega t}$ , where  $\omega = 2\pi f$  ( $f$  is the frequency) and  $\delta G^{\text{opt}}$  is an amplitude of a local perturbation.

The resulting small-signal device current perturbation  $\delta I_{\text{dev}}$  is the sum of real and imaginary parts, and the expressions for the quantum efficiency are:

$$\begin{aligned}\eta &= \frac{Re[\delta I_{\text{dev}}]/q}{\delta P^{\text{tot}}\lambda/hc} \\ C_{\text{opt}} &= \frac{1}{\omega} \cdot \frac{Im[\delta I_{\text{dev}}]/q}{\delta P^{\text{tot}}\lambda/hc} \\ \delta P^{\text{tot}} &= \int_S \delta P_0 ds\end{aligned}\tag{775}$$

where the quantity  $\delta P^{\text{tot}}\lambda/hc$  gives a perturbation of the total number of photons and  $Re[\delta I_{\text{dev}}]/q$  is a perturbation of the total number of electrons at an electrode. As a result, for each electrode, Sentaurus Device places two values into the AC output file, `photo_a` and `photo_c`, that correspond to  $\eta$  and  $C_{\text{opt}}$ , respectively. To start the optical AC analysis, add the keyword `Optical` in the `ACCoupled` statement. For example:

```
ACCoupled ( StartFrequency=1.e4 EndFrequency=1.e9  
    NumberOfPoints=31 Decade Node(a c) Optical )  
    { poisson electron hole }
```

#### Note:

If an element is excluded (`Exclude` statement) in optical AC (this is usually the case for voltage sources in regular AC simulation), it means that this element is not present in the simulated circuit and, correspondingly, it provides zero AC current for all branches that are connected to the element. Therefore, do *not* exclude voltage sources.

---

## References

- [1] B. R. Bennett, R. A. Soref, and J. A. Del Alamo, "Carrier-Induced Change in Refractive Index of InP, GaAs, and InGaAsP," *IEEE Journal of Quantum Electronics*, vol. 26, no. 1, pp. 113–122, 1990.
- [2] D. A. Clugston and P. A. Basore, "Modelling Free-carrier Absorption in Solar Cells," *Progress in Photovoltaics: Research and Applications*, vol. 5, no. 4, pp. 229–236, 1997.
- [3] D. K. Schroder, R. N. Thomas, and J. C. Swartz, "Free Carrier Absorption in Silicon," *IEEE Journal of Solid-State Circuits*, vol. SC-13, no. 1, pp. 180–187, 1978.
- [4] H. E. Bennett and J. O. Porteus, "Relation Between Surface Roughness and Specular Reflectance at Normal Incidence," *Journal of the Optical Society of America*, vol. 51, no. 2, pp. 123–129, 1961.
- [5] P. Beckmann and A. Spizzichino, *The Scattering of Electromagnetic Waves from Rough Surfaces*, Norwood, Massachusetts: Artech House, 1987.
- [6] J. Krc, F. Smole, and M. Topic, "Analysis of Light Scattering in Amorphous Si:H Solar Cells by a One-Dimensional Semi-coherent Optical Model," *Progress in Photovoltaics: Research and Applications*, vol. 11, no. 1, pp. 15–26, 2003.
- [7] M. D. Feit and J. A. Fleck, Jr., "Light propagation in graded-index optical fibers," *Applied Optics*, vol. 17, no. 24, pp. 3990–3998, 1978.
- [8] P. Kaczmarski and P. E. Lagasse, "Bidirectional Beam Propagation Method," *Electronics Letters*, vol. 24, no. 11, pp. 675–676, 1988.
- [9] F. Alauzet and M. Mehrenberger, " $P^1$ -conservative solution interpolation on unstructured triangular meshes," *International Journal for Numerical Methods in Engineering*, vol. 84, no. 13, pp. 1552–1588, 2010.

# 22

## Radiation

---

*This chapter describes the radiation models used in Sentaurus Device.*

When high-energy particles penetrate a semiconductor device, they deposit their energy by the generation of electron–hole pairs. These charges can perturb the normal operation of the device. This chapter describes the models for carrier generation by gamma radiation, alpha particles, and heavy ions.

---

### Carrier Generation by Gamma Radiation

To activate the gamma radiation model, specify the keyword `Radiation(...)`, with optional parameters, in the `Physics` section. For example:

```
Physics { ...
    Radiation(
        Dose = <float> | DoseRate = <float>
        DoseTime = (<float>,<float>)
        DoseTSigma = <float>
    )
}
```

where:

- `DoseRate` (in rad/s) represents  $D$  in [Equation 776](#).
- The optional `DoseTime` (in s) allows you to specify the time period during which exposure to the constant `DoseRate` occurs.
- `DoseTSigma` (in s) can be combined with `DoseTime` to specify the standard deviation of a Gaussian rise and fall of the radiation exposure.
- As an alternative to `DoseRate`, you can specify `Dose` (in rad) to represent the total radiation exposure over the `DoseTime` interval. In this case, `DoseTime` must be specified.

To plot the generation rate due to gamma radiation, specify `RadiationGeneration` in the `Plot` section.

---

## Yield Function

Generation of electron–hole pairs due to radiation is an electric field–dependent process [1] and is modeled as follows:

$$G_r = g_0 D \cdot Y(F)$$
$$Y(F) = \frac{F + E_0}{F + E_1}^m \quad (776)$$

where  $D$  is the dose rate,  $g_0$  is the generation rate of electron–hole pairs, and  $E_0$ ,  $E_1$ , and  $m$  are constants.

All these constants can be specified in the parameter file of Sentaurus Device as follows:

```
Radiation {  
    g = 7.6000e+12      # [1/(rad*cm^3)]  
    E0 = 0.1            # [V/cm]  
    E1 = 1.3500e+06    # [V/cm]  
    m = 0.9            # [1]  
}
```

---

## Carrier Generation by Alpha Particles

To activate the alpha particle model, specify the keyword `AlphaParticle(...)`, with optional parameters, in the `Physics` section. For example:

```
Physics { ...  
    AlphaParticle (<keyword_options>)  
}
```

[Table 282 on page 1668](#) lists the keyword options for alpha particles.

To plot the instant generation rate  $G(t)$  due to alpha particles, specify `AlphaGeneration` in the `Plot` section. To plot  $\sum_{\infty}^{\infty} dG$ , specify `AlphaCharge`.

**Note:**

The generation by alpha particles cannot be used except in transient simulations.  
The number of electron–hole pairs generated before the initial time of the transient is added to the carrier densities at the beginning of the simulation.

An option to improve the spatial integration of the charge generation is presented in [References on page 780](#).

## Chapter 22: Radiation

### Carrier Generation by Alpha Particles

## Using the Alpha Particle Model

The generation rate caused by an alpha particle with energy  $E$  is computed by [2]:

$$G(u, v, w, t) = \frac{a}{\sqrt{2\pi} \cdot s} \exp\left[-\frac{1}{2} \frac{(t - t_m)^2}{s^2} - \frac{1}{2} \frac{v^2 + w^2}{w_t^2}\right] \left[ c_1 e^{\alpha u} + c_2 \exp\left(-\frac{1}{2} \frac{u - \alpha_1}{\alpha_2}\right)^2 \right] \quad (777)$$

if  $u < \alpha_1 + \alpha_3$ , and by:

$$G(u, v, w, t) = 0 \quad (778)$$

if  $u \geq \alpha_1 + \alpha_3$ . In this case,  $u$  is the coordinate along the particle path, and  $v$  and  $w$  are coordinates orthogonal to  $u$ . The direction and place of incidence are defined in the Physics section of the command file with the keywords `Direction` and `Location` (or `StartPoint`, see [Note on page 778](#)), respectively. The  $t_m$  parameter is the time of the generation peak defined by the keyword `Time`. A Gaussian time dependence can also be used to simulate the typical generation due to pulsed laser or electron beams.

The maximum of the Bragg peak,  $\alpha_1$ , is fitted to data [3] by a polynomial function:

$$\alpha_1 = a_0 + a_1 E + a_2 E^2 \quad (779)$$

The parameter  $c_1$  is given by:

$$c_1 = \exp[\alpha(\alpha_1[10\text{MeV}] - \alpha_1[E])] \quad (780)$$

The scaling factor  $a$  is determined from:

$$\sum_{0}^{\infty} \sum_{-\infty}^{\infty} \sum_{-\infty}^{\infty} G(u, v, w, t) dt dw dv du = \frac{E}{E_p} \quad (781)$$

where  $E_p$  is the average energy needed to create an electron–hole pair. The remaining parameters are listed in [Table 137](#). They are available in the parameter set `AlphaParticle` and are valid for alpha particles with energies between 1 MeV and 10 MeV.

*Table 137 Coefficients for carrier generation by alpha particles*

Symbol	Parameter name	Default value	Unit
$s$	s	$2 \times 10^{-12}$	s
$w_t$	wt	$1 \times 10^{-5}$	cm
$c_2$	c2	1.4	1
$a$	alpha	90	$\text{cm}^{-1}$
$\alpha_2$	alpha2	$5.5 \times 10^{-4}$	cm
$\alpha_3$	alpha3	$2 \times 10^{-4}$	cm

## Chapter 22: Radiation

### Carrier Generation by Heavy Ions

Table 137 Coefficients for carrier generation by alpha particles (Continued)

Symbol	Parameter name	Default value	Unit
$E_p$	Ep	3.6	eV
$a_o$	a0	$-1.033 \times 10^{-4}$	cm
$a_1$	a1	$2.7 \times 10^{-10}$	cm/eV
$a_2$	a2	$4.33 \times 10^{-17}$	cm/eV <sup>2</sup>

---

## Carrier Generation by Heavy Ions

When a heavy ion penetrates a device structure, it loses energy and creates a trail of electron–hole pairs. These additional electrons and holes might cause a large enough current to switch the logic state of a device, for example, a memory cell. Important factors are:

- The energy and type of the ion
- The angle of penetration of the ion
- The relation between the lost energy or linear energy transfer (LET) and the number of pairs created

The simulation of a single event upset caused by a heavy ion impact is activated by using the keyword `HeavyIon` in an appropriate `Physics` section. For example:

```
# Default heavy ion
Physics {
    HeavyIon (<keyword_options>)
}
# User-defined heavy ion
Physics {
    HeavyIon (<IonName>) (<keyword_options>)
}
```

### Note:

User-defined heavy ions do not have default parameters. Therefore, in the parameter file, a specification of the following form must appear (see [Table 139 on page 778](#)):

```
HeavyIon(<IonName>) ( . . . )
```

[Table 283 on page 1668](#) describes the options for `HeavyIon`. The generation rate by the heavy ion is generally used in transient simulations. The number of electron–hole pairs generated before the initial time of the transient is added to the carrier densities at the beginning of the simulation. The total charge density and the instant generation rate are

## Chapter 22: Radiation

### Carrier Generation by Heavy Ions

plotted using `HeavyIonChargeDensity` and `HeavyIonGeneration` in the `Plot` section, respectively.

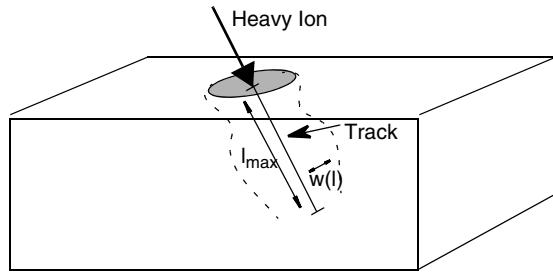
If the value of `wt_hi` is 0, then uniform generation is selected. If the value of `LET_f` is 0, then the keyword `LET_f` can be ignored.

---

## Using the Heavy Ion Model

[Figure 49](#) shows a simple model for the heavy ion impinging process.

[Figure 49](#) A heavy ion penetrating a semiconductor; its track is defined by a length and the transverse spatial influence is assumed to be symmetric about the track axis



The generation rate caused by the heavy ion is computed by:

$$G(l, w, t) = G_{\text{LET}}(l)R(w, l)T(t) \quad (782)$$

if  $l < l_{\max}$  ( $l_{\max}$  is the length of the track), and by:

$$G(l, w, t) = 0 \quad (783)$$

if  $l \geq l_{\max}$ .  $R(w)$  and  $T(t)$  are functions describing the spatial and temporal variations of the generation rate.  $G_{\text{LET}}(l)$  is the LET generation density and its unit is pairs/cm<sup>3</sup>.

$T(t)$  is defined as a Gaussian function:

$$T(t) = \frac{2 \cdot \exp \left( -\frac{(t - t_0)^2}{2 \cdot s_{\text{hi}}^2} \right)}{\sqrt{2 \cdot s_{\text{hi}}} \cdot \sqrt{\pi} \cdot \left[ 1 + \operatorname{erf} \left( \frac{t_0}{\sqrt{2 \cdot s_{\text{hi}}}} \right) \right]} \quad (784)$$

where  $t_0$  is the moment of the heavy ion penetration (see the keyword `Time` in [Table 283 on page 1668](#)), and  $s_{\text{hi}}$  is the characteristic value of the Gaussian (see `s_hi` in [Table 139 on page 778](#)).

## Chapter 22: Radiation

### Carrier Generation by Heavy Ions

The spatial distribution,  $R(w, l)$ , can be defined as either an exponential function (default):

$$R(w, l) = \exp -\frac{w}{w_t(l)} \quad (785)$$

or a Gaussian function:

$$R(w, l) = \exp -\frac{w^2}{w_t(l)^2} \quad (786)$$

where  $w$  is a radius defined as the perpendicular distance from the track. The characteristic distance  $w_t$  is defined as `wt_hi` in the `HeavyIon` statement and can be a function of the length  $l$  (see [Table 283 on page 1668](#)).

In addition, the spatial distribution  $R(w, l)$  can be defined as a PMI function (see [Heavy Ion Spatial Distribution on page 1462](#)).

The LET generation density,  $G_{\text{LET}}(l)$ , is given by:

$$G_{\text{LET}}(l) = a_1 + a_2 l + a_3 e^{a_4 l} + k' \left[ c_1 (c_2 + c_3 l)^{c_4} + \text{LET\_f}(l) \right] \quad (787)$$

where `LET_f(l)` (defined by the keyword `LET_f`) is a function of the length  $l$ . [Example 2 on page 779](#) illustrates how to use an array of values to specify the length dependence of `LET_f(l)`. A linear interpolation is used for values between the array entries of `LET_f`.

There are two options for the units of `LET_f`: pairs/cm<sup>3</sup> (default) or pC/μm (activated by the keyword `PicoCoulomb`). Depending on the units chosen,  $k'$  takes on different values in order to make [Equation 787](#) dimensionally consistent. [Table 138](#) summarizes the appropriate values of  $k'$  for different device dimensions.

*Table 138 Setting correct  $k'$  to make LET generation density equation dimensionally consistent*

Condition	Two-dimensional device	Three-dimensional device
<code>LET_f</code> has units of pairs/cm <sup>3</sup> and $R(w, l)$ is exponential or Gaussian	$k' = k$	$k' = k$
<code>LET_f</code> has units of pC/μm and $R(w, l)$ is exponential	$k' = \frac{k}{2w_t d}$ $d = 1 \mu\text{m}$	$k' = \frac{k}{2\pi w_t^2}$
<code>LET_f</code> has units of pC/μm and $R(w, l)$ is Gaussian	$k' = \frac{k}{\sqrt{\pi} w_t d}$ $d = 1 \mu\text{m}$	$k' = \frac{k}{\pi w_t^2}$

## Chapter 22: Radiation

### Carrier Generation by Heavy Ions

Exercise care when choosing the correct units for `wt_hi` and `Length`. The default unit of `wt_hi` and `Length` is centimeter. If you specify `PicoCoulomb`, then the unit becomes  $\mu\text{m}$ . For examples illustrating the correct unit use, see [Examples: Heavy Ions](#).

[Table 139](#) lists the other coefficients used in [Equation 787](#) with their default values, and they can be adjusted in the parameter file of Sentaurus Device.

#### Note:

The keyword `Location` defines a bidirectional track for which Sentaurus Device computes the generation rate in both directions from the place of incidence along the `Direction` vector. The keyword  `startPoint` defines a one-directional track. In this case, Sentaurus Device computes the generation rate only in the positive direction from the place of incidence.

*Table 139 Coefficients for carrier generation by heavy ion (HeavyIon parameter set)*

	<code>s_hi</code>	<code>a<sub>1</sub></code>	<code>a<sub>2</sub></code>	<code>a<sub>3</sub></code>	<code>a<sub>4</sub></code>	<code>k</code>	<code>c<sub>1</sub></code>	<code>c<sub>2</sub></code>	<code>c<sub>3</sub></code>	<code>c<sub>4</sub></code>
<b>Keyword</b>	<code>s_hi</code>	<code>a_1</code>	<code>a_2</code>	<code>a_3</code>	<code>a_4</code>	<code>k_hi</code>	<code>c_1</code>	<code>c_2</code>	<code>c_3</code>	<code>c_4</code>
<b>Default value</b>	2e-12	0	0	0	0	1	0	1	0	1
<b>Default unit</b>	s	pairs/cm <sup>3</sup>	pairs/cm <sup>3</sup> /cm	pairs/cm <sup>3</sup>	cm <sup>-1</sup>	1	pairs/cm <sup>3</sup>	1	cm <sup>-1</sup>	1
<b>Unit if PicoCoulomb is chosen</b>	s	pairs/cm <sup>3</sup>	pairs/cm <sup>3</sup> / $\mu\text{m}$	pairs/cm <sup>3</sup>	$\mu\text{m}^{-1}$	1	pC/ $\mu\text{m}$	1	$\mu\text{m}^{-1}$	1

## Examples: Heavy Ions

This section presents examples of using the heavy ion model.

### Example 1

The track has a constant `LET_f` value of 0.2 pC/ $\mu\text{m}$  across the track. The track length is 1  $\mu\text{m}$  ( $l_{\max} = 1 \mu\text{m}$ ) and the heavy ion crosses the device at time 0.1 ps. The unit of `LET_f` is pC/ $\mu\text{m}$  and the spatial distribution is Gaussian. Since `PicoCoulomb` is specified, the values of `Length` and `wt_hi` are expressed in terms of  $\mu\text{m}$ . The keyword `HeavyIonChargeDensity` in the `Plot` section plots the charge density generated by the ion.

```
Physics {
    Recombination ( SRH(DopingDependence) )
    Mobility (DopingDependence Enormal HighFieldSaturation)
    HeavyIon (
        Direction=(0,1)
        Location=(1.5,0)
```

## Chapter 22: Radiation

### Carrier Generation by Heavy Ions

```
Time=1.0e-13
Length=1
Wt_hi=3
LET_f=0.2
Gaussian
PicoCoulomb
)
}
Plot { eDensity hDensity ElectricField HeavyIonChargeDensity }
```

## Example 2

The `LET_f` and radius (`Wt_hi`) values are functions of the position along the track (in this case,  $l_{\max} = 1.7 \times 10^{-4}$  cm). Values in between the array entries are interpolated linearly. The unit of `LET_f` is pairs/cm<sup>3</sup> (because `PicoCoulomb` is not specified), and the unit for `Length` and `Wt_hi` is centimeter. For each value of length, there is a corresponding value of `LET_f` and a value for the radius. The spatial distribution in the perpendicular direction from the track is exponential.

```
Physics {
    Recombination ( SRH(DopingDependence) )
    HeavyIon (
        Direction=(0,1)
        Location=(1.5,0)
        Time=1.0e-13
        Length = [1e-4 1.5e-4 1.6e-4 1.7e-4]
        LET_f = [1e6 2e6 3e6 4e6]
        Wt_hi = [0.3e-4 0.2e-4 0.25e-4 0.1e-4]
        Exponential
    )
}
```

## Example 3

This example illustrates multiple ion strikes in the model.

```
Physics {
    Recombination ( SRH(DopingDependence) )
    HeavyIon (
        Direction=(0,1)
        Location=(0,0)
        Time=1.0e-13
        Length = [1e-4 1.5e-4 1.6e-4 1.7e-4]
        LET_f = [1e6 2e6 3e6 4e6]
        Wt_hi = [0.3e-4 0.2e-4 0.25e-4 0.1e-4]
    )
    HeavyIon ("Ion1")(
        Direction=(0,1)
        Location=(1,0)
        Time=1.0e-13
        Length = [1e-4 1.5e-4 1.6e-4 1.7e-4]
```

## Chapter 22: Radiation

### References

```
    LET_f = [1e6 2e6 3e6 4e6]
    Wt_hi = [0.3e-4 0.2e-4 0.25e-4 0.1e-4]
)
}
```

User-defined heavy ions do not have default parameters. Therefore, in the parameter file, a specification of the following form must appear (see [Table 139 on page 778](#)):

```
HeavyIon("Ion1")(...)
```

---

## References

- [1] J.-L. Leray, "Total Dose Effects: Modeling for Present and Future," *IEEE Nuclear and Space Radiation Effects Conference (NSREC) Short Course*, 1999.
- [2] A. Erlebach, *Modellierung und Simulation strahlensensitiver Halbleiterbauelemente*, Aachen: Shaker, 1999.
- [3] L. C. Northcliffe and R. F. Schilling, "Range and Stopping - Power Tables for Heavy Ions," *Nuclear Data Tables*, vol. A7, no. 1–2, New York: Academic Press, pp. 233–463, 1969.

# 23

## Noise, Fluctuations, and Sensitivity

---

*This chapter discusses noise analysis, fluctuation analysis, and sensitivity analysis in Sentaurus Device.*

This chapter explains the Sentaurus Device features to model noise, statistical fluctuations of doping, trap concentration, workfunction, band edge, geometry, dielectric constant, metal conductivity, and sensitivity to variations of model parameters, doping, and geometry. All these topics deal with the response of the device characteristics to small, device-internal variations. For noise, these variations occur randomly over time within a single device; for statistical fluctuations, the variations are random device-to-device variations; and for sensitivity, the variations are user supplied. Despite the different nature of the variations, they all can be handled by the same linearization approach called the *impedance field method* and, therefore, Sentaurus Device treats them all in a common framework.

[Using the Impedance Field Method](#) explains how to perform noise, fluctuation, and sensitivity analysis. [Noise Sources on page 787](#) discusses the noise and random fluctuation models that Sentaurus Device offers. [Statistical Impedance Field Method on page 797](#) describes a modeling approach for random dopant fluctuations, trap concentration fluctuations, geometric variations, and workfunction variations based on statistical sampling. [Deterministic Variations on page 808](#) discusses user-supplied variations of doping, geometry, and model parameters.

[Impedance Field Method on page 814](#) discusses the background of the method [1], and [Noise Output Data on page 815](#) summarizes the data available for visualization.

---

### Using the Impedance Field Method

Sentaurus Device treats noise analysis, fluctuation analysis, and sensitivity analysis by the impedance field method, and as extensions of small-signal analysis (see [Small-Signal AC Analysis on page 149](#)).

## Chapter 23: Noise, Fluctuations, and Sensitivity

Using the Impedance Field Method

### Note:

The impedance field method is not supported with the `HeteroInterface` option (see [Abrupt and Graded Heterojunctions on page 59](#)), the `Thermionic` option (see [Thermionic Emission Current on page 870](#)), or dipole layers ([Dipole Layer on page 230](#)).

For noise and random fluctuations, Sentaurus Device computes the variances and correlation coefficients for the voltages at selected circuit nodes, assuming the net current to these nodes is fixed. As the computation is performed in frequency space, the computed quantities are called the noise voltage spectral densities. Sentaurus Device also computes the variances and correlation coefficients of the currents through the nodes, assuming fixed voltages; these quantities are the noise current spectral densities.

For the statistical impedance field method (sIFM) and for deterministic variations, Sentaurus Device computes responses of node voltages assuming fixed currents and responses of node currents assuming fixed voltages.

---

## Specifying Variations

To use noise and random fluctuation analysis, specify the physical models for the microscopic origin of the deviations (called the local noise sources, LNS) as options to the keyword `Noise` in the `Physics` section of the command file of Sentaurus Device:

```
Physics {  
    ...  
    Noise <string> ( <Noisemodels> )  
}
```

where `<Noisemodels>` is a list that specifies any number of noise models listed in [Table 319 on page 1692](#). The name given by the string after `Noise` is optional. Using `Noise` specifications with different names allows you to investigate different specifications of noise in a single simulation run.

For the sIFM, specify `RandomizedVariation` in the global `Math` section and in a device-specific global `Physics` section:

```
Math {  
    RandomizedVariation <string> ( <specification> )  
    ...  
}  
Device <string> {  
    Physics {  
        RandomizedVariation <string> ( <specification> )  
    }  
}
```

where `<specification>` specifies the details for the randomization (see [Statistical Impedance Field Method on page 797](#)).

## Chapter 23: Noise, Fluctuations, and Sensitivity

Using the Impedance Field Method

For deterministic variations, specify the variations as options to `DeterministicVariation` in the global `Physics` section:

```
Physics {
    DeterministicVariation( <variations> )
    ...
}
```

where `<variations>` is a list that specifies any number of deterministic variations (see [Deterministic Variations on page 808](#) and [Table 290 on page 1672](#)).

---

## Specifying the Solver

In any case, use the `ObservationNode` option to the `ACCoupled` statement to specify the device nodes for which the noise spectral densities or deviations are required. For example:

```
ACCoupled (
    StartFrequency = 1.e8 EndFrequency = 1.e11
    NumberOfPoints = 7 Decade
    Node (n_source n_drain n_gate)
    Exclude (v_drain v_gate)
    ObservationNode (n_drain n_gate)
    ACEExtract = "mos"
    NoisePlot = "mos"
) {
    poisson electron hole contact circuit
}
```

The keyword `ObservationNode` enables noise and fluctuation analysis (in this case, for the nodes `n_drain` and `n_gate`). `NoisePlot` specifies a file name prefix for device-specific plots (see [Noise Output Data on page 815](#)). For more information on the `ACCoupled` statement, see [Small-Signal AC Analysis on page 149](#).

### Note:

The observation nodes must be a subset of the nodes specified in `Node(...)`.

---

## Analysis at Frequency Zero

You can perform variation analysis at frequency zero. If you specify in the `ACCoupled` statement the single frequency zero by:

```
StartFrequency = 0. EndFrequency = 0. NumberOfPoints = 1
```

this mode is activated, which enhances both speed and memory consumption of the analysis, compared to the analysis at positive frequency.

## Chapter 23: Noise, Fluctuations, and Sensitivity

Using the Impedance Field Method

However, you must be aware of the following:

- Capacitances of AC nodes cannot be extracted and are set to zero in the output files.
- For general structures, only the current Green's functions (and their responses) are well defined and computed. The voltage Green's functions and their related responses cannot be computed. Therefore, their computation is deactivated by default and corresponding output data is set to zero. However, if all parts of the structure are conductively (and not only capacitively) coupled, then voltage responses are well defined and their computation can be activated explicitly by specifying `VoltageGreenFunctions` as a parameter in the `ACCoupled` statement.

---

## Output of Results

The results of the analysis are the noise voltage spectral densities, the noise current spectral densities, and voltage and current deviations. They appear in the `ACExtract` file (see [Table 140](#)) and, in the case of the sIFM, in a separate `.csv` file. The units are  $V^2$ 's for the voltage spectral densities and  $A^2$ 's for the current noise spectral densities. If the string specified after `Noise` is not empty, it is prefixed to the name of the spectral density in the `ACExtract` file.

*Table 140     Noise and fluctuation data written into ACExtract file*

Name	Description
S_V	Autocorrelation noise voltage spectral density (NVSD)
S_I	Autocorrelation noise current spectral density (NISD)
S_V_ee	Electron NVSD
S_V_hh	Hole NVSD
S_V_eeDiff	Electron NVSD due to diffusion LNS
S_V_hhDiff	Hole NVSD due to diffusion LNS
S_V_eeMonoGR	Electron NVSD due to monopolar GR LNS
S_V_hhMonoGR	Hole NVSD due to monopolar GR LNS
S_V_eeFlickerGR	Electron NVSD due to Flicker GR LNS
S_V_hhFlickerGR	Hole NVSD due to Flicker GR LNS
S_V_BandEdge	NVSD and NISD due to band edge fluctuations
S_I_BandEdge	

## Chapter 23: Noise, Fluctuations, and Sensitivity

Using the Impedance Field Method

Table 140 Noise and fluctuation data written into ACExtract file (Continued)

Name	Description
S_V_Conductivity	NVSD and NISD due to metal conductivity fluctuations
S_I_Conductivity	
S_V_Doping	NVSD and NISD due to random dopant fluctuations
S_I_Doping	
S_V_Epsilon	NVSD and NISD due to dielectric constant variations
S_I_Epsilon	
S_V_Geometric	NVSD and NISD due to geometric fluctuations
S_I_Geometric	
S_V_TrapConcentration	NVSD and NISD due to trap concentration fluctuations
S_I_TrapConcentration	
S_V_Trap	NVSD and NISD due to trapping noise
S_I_Trap	
S_V_Workfunction	NVSD and NISD due to workfunction fluctuations
S_I_Workfunction	
ReS_VXV	Real/imaginary parts of the cross correlation noise voltage spectral density (NVXVSD)
ImS_VXV	
ReS_IXI	Real/imaginary parts of the cross correlation noise current spectral density
ImS_IXI	
ReS_VXV_ee	Real/imaginary parts of the electron/hole (NVXVSD)
ImS_VXV_ee	
ReS_VXV_hh	
ImS_VXV_hh	
ReS_VXV_eeDiff	Real/imaginary parts of the electron/hole NVXVSD due to diffusion LNS
ImS_VXV_eeDiff	
ReS_VXV_hhDiff	
ImS_VXV_hhDiff	
ReS_VXV_eeMonoGR	Real/imaginary parts of the electron/hole NVXVSD due to monopolar GR LNS
ImS_VXV_eeMonoGR	
ReS_VXV_hhMonoGR	
ImS_VXV_hhMonoGR	

## Chapter 23: Noise, Fluctuations, and Sensitivity

Using the Impedance Field Method

*Table 140 Noise and fluctuation data written into ACExtract file (Continued)*

Name	Description
ReS_VXV_eeFlickerGR	Real/imaginary parts of the electron/hole NVXVSD due to Flicker GR LNS
ImS_VXV_eeFlickerGR	
ReS_VXV_hhFlickerGR	
ImS_VXV_hhFlickerGR	
ReS_VXV_BandEdge	Real part of the NVXVSD and NIXISD due to band edge fluctuations
ReS_IXI_BandEdge	
ReS_VXV_Conductivity	Real part of the NVXVSD and NIXISD due to metal conductivity fluctuations
ReS_IXI_Conductivity	
ReS_VXV_Doping	Real part of the NVXVSD and NIXISD due to random dopant fluctuations
ReS_IXI_Doping	
ReS_VXV_Epsilon	Real part of the NVXVSD and NIXISD due to dielectric constant fluctuations
ReS_IXI_Epsilon	
ReS_VXV_Geometric	Real part of the NVXVSD and NIXISD due to geometric fluctuations
ReS_IXI_Geometric	
ReS_VXV_TrapConcentration	Real part of the NVXVSD and NIXISD due to trap concentration fluctuations
ReS_IXI_TrapConcentration	
ReS_VXV_Trap	Real/imaginary parts of the NVXVSD and NIXISD due to trapping noise
ImS_VXV_Trap	
ReS_IXI_Trap	
ImS_IXI_Trap	
ReS_VXV_Workfunction	Real parts of the NVXVSD and NIXISD due to workfunction fluctuations
ReS_IXI_Workfunction	
dV<name>	Voltage deviation due to deterministic variation <name>
dI<name>	Current deviation due to deterministic variation <name>

## Noise Sources

**Note:**

Noise scales differently with the device width compared to currents or voltages. For all noise sources noted in this section, for 2D devices, the device width is assumed to be given by `AreaFactor`. Therefore, it is not necessary to perform a full 3D simulation to obtain the correct scaling behavior for an essentially 2D structure. For 3D structures, when simulating only half (or a quarter) of the structure, you can use `AreaFactor=2` (or `AreaFactor=4`) to obtain the correct scale for the full structure, provided that spatial correlations of the noise sources can be neglected.

---

## Common Options

All noise sources discussed in this section can be specified with the parameters `SpaceMid`, `SpaceSig`, and `SpatialShape`. These parameters can restrict the noise source to a window. The possible values, the default values, and the modifier function specified by these keywords are described in [Energetic and Spatial Distribution of Traps on page 544](#). The modifier function is applied to each of the two coordinates on which a noise source depends. For noise sources without spatial correlation, this means that the square of the modifier function is multiplied by the noise source.

For example, the following unnamed `Noise` section activates random dopant fluctuations globally:

```
Noise (Doping)
```

The following `Noise` section named `window` activates random dopant fluctuations in a cube of 20 nm side length, centered at the origin:

```
Noise "window" (
    Doping (
        SpaceMid = (0 0 0)
        SpaceSig = (0.01 0.01 0.01)
        SpatialShape = Uniform
    )
)
```

---

## Diffusion Noise

The diffusion noise source (keyword `DiffusionNoise`) available in Sentaurus Device reads:

$$K_{n,n}^{\text{Diff}}(\vec{r}_1, \vec{r}_2, \omega) = 4qn(\vec{r}_1)kT_n(\vec{r}_1)\mu_n(\vec{r}_1)\delta(\vec{r}_1 - \vec{r}_2) \quad (788)$$

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

A corresponding expression is used for holes.  $K_{n,n}^{\text{Diff}}$  is a diagonal tensor.

$T_n$  is either the lattice or carrier temperature, depending on the specification in the command file:

```
DiffusionNoise ( <temp_option> )
```

where `<temp_option>` is `LatticeTemperature` (default), `eTemperature`, `hTemperature`, or `e_h_Temperature`.

For example, if the following command is specified:

```
Physics {  
    Noise ( DiffusionNoise ( eTemperature ) )  
}
```

Sentaurus Device uses the electron temperature for the electron noise source and the lattice temperature for the hole noise source. The keyword `e_h_Temperature` forces the corresponding carrier temperature to be used for the diffusion noise source for each carrier type.

---

## Equivalent Monopolar Generation–Recombination Noise

An equivalent monopolar generation–recombination (GR) noise source model (keyword `MonopolarGRNoise`) for a two-level, GR process can be expressed as a tensor [2]:

$$K_{n,n}^{\text{GR}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\vec{J}_n(\vec{r}_1)\vec{J}_n(\vec{r}_2)}{n} \cdot \frac{4\alpha\tau_{\text{eq}}}{1 + \omega^2\tau_{\text{eq}}^2} \delta(\vec{r}_1 - \vec{r}_2) \quad (789)$$

where  $\alpha$  is a fitting parameter and  $\tau_{\text{eq}}$  an equivalent GR lifetime. The parameters  $\tau_{\text{eq}}$  and  $\alpha$  can be modified in the parameter file of Sentaurus Device. A similar expression is used for holes.

### Note:

This model does not use the actual generation–recombination models activated in the simulation. Therefore, it cannot be considered a physical model for recombination noise.

---

## Bulk Flicker Noise

The flicker generation–recombination (GR) noise model (keyword `FlickerGRNoise`) for electrons (similar for holes) is:

$$K_{n,n}^{\text{fGR}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\vec{J}_n(\vec{r}_1)\vec{J}_n(\vec{r}_2)}{n(\vec{r}_1)} \frac{2\alpha_H}{\pi v \ln(\tau_1/\tau_0)} [\tan(\omega\tau_1) - \tan(\omega\tau_0)] \delta(\vec{r}_1 - \vec{r}_2) \quad (790)$$

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

where  $\alpha_H$  is a fit parameter;  $\omega = 2\pi\nu$ , the angular frequency; and the time constants fulfill  $\tau_0 < \tau_1$ . The parameters  $\alpha_H$ ,  $\tau_0$ , and  $\tau_1$  for electrons and holes are accessible in the parameter file. With increasing frequency, the noise source changes from constant to  $1/\nu^2$  behavior close to the frequency  $\nu_1 = 1/\tau_1$  and, ultimately, to  $1/\nu^3$  behavior at  $\nu_0 = 1/\tau_0$ .

#### Note:

This model does not use the actual generation–recombination and trap models activated in the simulation. Therefore, it cannot be considered a physical model for flicker noise.

---

## Trapping Noise

The Traps option to Noise activates a trapping noise model that follows the microscopic model in [3]. The trapping noise sources are determined fully by the trap model and, therefore, do not require additional adjustable parameters. For example:

```
Physics { Noise(Traps) }
```

activates trapping noise for all traps in the device. It is not possible to activate trapping noise for selected traps only. All trap models apart from tunneling to electrodes are supported. For more details on traps, see [Chapter 17 on page 543](#).

Consider a trap level  $k$  with concentration  $N_{t,k}$  and trap electron occupation  $f_k$ . Trapping noise is expressed by adding a Langevin source  $s_{i,k}$  to the net electron capture rate for each process  $i$ :

$$R_{i,k} = N_{t,k}(1-f_k)c_{i,k} - N_{t,k}f_k e_{i,k} + s_{i,k} \quad (791)$$

where  $c_{i,k}$  is the electron capture rate for a trap occupied by zero electrons, and  $e_{i,k}$  is the electron emission rate for a trap occupied by one electron (see also [Equation 526 on page 553](#)). Denoting the expectation value with  $\langle \rangle$ , the trapping noise source takes the form:

$$K_{ij,k}^{\text{trap}}(\omega) = 2q^2 \int dt s_{i,k}(0)s_{j,k}(t) \exp(-i\omega t) = 2q^2 \delta_{ij} N_{t,k} [(1-f_k)c_{i,k} + f_k e_{i,k}] \quad (792)$$

That is, different capture or emission processes are independent, and the noise source for a given process is twice the total trapping rate (the sum of the capture and emission rates). The model also assumes that different trap distributions are independent, so that their noise contributions add.

---

## Random Dopant Fluctuations

The noise sources for random dopant fluctuations are activated by the `Doping` keyword. The noise source for acceptor fluctuations reads:

$$K_A^{\text{RDF}}(\vec{r}_1, \vec{r}_2, \omega) = N_A(\vec{r}_1) \frac{\Theta(0.5\text{Hz} - |\nu|)}{1\text{Hz}} \delta(\vec{r}_1 - \vec{r}_2) \quad (793)$$

Here,  $\nu = \omega/2\pi$  is the frequency. An analogous expression holds for the noise source for donors,  $K_D^{\text{RDF}}$ . Physically, the noise sources are static. However, to avoid a  $\delta$ -function in frequency space, Sentaurus Device spreads the spectral density of the noise source over a 1 Hz frequency interval. [Equation 793](#) is based on the assumption that individual dopant atoms are completely uncorrelated.

Acceptors and donors are considered to be independent. Their contributions to the noise spectral densities add. By default, both acceptor and donor concentrations are assumed to fluctuate. Using either the option `Type=Acceptor` or `Type=Donor` of `Doping`, the fluctuation can be restricted to acceptors or donors, respectively. For example:

```
Physics {  
    Noise "acceptor" ( Doping(Type=Acceptor) )  
    Noise "donor" ( Doping(Type=Donor) )  
}
```

declares two named `Noise` sections that allow you to examine the impact of the fluctuation of acceptors and donors separately.

By default, Sentaurus Device neglects the impact of the random dopant fluctuations on mobility and bandgap narrowing. To take their impact into account, specify the `Mobility` and `BandgapNarrowing` options to the `Doping` keyword. For example:

```
Physics { Noise( Doping(Mobility) ) }
```

activates the random dopant fluctuation noise source, taking into account the impact of the fluctuations on mobility.

**Note:**

It is strongly advisable to use either both the `BandgapNarrowing` and `Mobility` options or neither option. Otherwise, inconsistencies can arise when the mobility depends indirectly on bandgap narrowing. Even when this dependency is weak, the inconsistency can lead to large errors in IFM results.

---

## Random Geometric Fluctuations

The geometric fluctuation model accounts for random displacements of electrodes on insulator, as well as metal–insulator, insulator–insulator, and semiconductor–insulator interfaces.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

The noise source reads:

$$K^{\text{geo}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\Theta(0.5\text{Hz} - |\nu|)}{1\text{Hz}} \delta s(\vec{r}_1) \delta s(\vec{r}_2) \quad (794)$$

where  $\nu = \omega/2\pi$  is the frequency, and the correlation of the displacements  $\delta s$  is given by:

$$\delta s(\vec{r}_1) \delta s(\vec{r}_2) = \hat{n}(\vec{r}_1) \cdot \hat{n}(\vec{r}_2) [a_{\text{iso}}(\vec{r}_1) + |\hat{a}(\vec{r}_1) \cdot \hat{n}(\vec{r}_1)|] [a_{\text{iso}}(\vec{r}_2) + |\hat{a}(\vec{r}_2) \cdot \hat{n}(\vec{r}_2)|] \exp\left[-\frac{(\vec{r}_1 - \vec{r}_2)^2}{\lambda^2}\right] \quad (795)$$

where:

- $\vec{r}_1$  and  $\vec{r}_2$  are positions on the interface.
- $\hat{n}(\vec{r})$  is the local interface normal in point  $\vec{r}$ .
- $a_{\text{iso}}$  (the isotropic correlation amplitude),  $\hat{a}$  (the vectorial correlation amplitude), and  $\lambda$  (the correlation length) are adjustable parameters.

The correlation has the following noteworthy properties:

- Displacements of interface positions occur only in the interface normal direction.
- The displacement amplitude depends on the interface direction when  $\hat{a}$  is nonzero.
- Displacements are spatially correlated; the correlations decay with increasing distance.
- Correlations depend on the relative normal direction. Perpendicular interfaces are uncorrelated; opposing interfaces are negatively correlated (so the actual shifts are positively correlated).

The impact of displacements is accounted for in the variation of the dielectric constant in the Poisson equation, in the variation of the space charge in the Poisson equation, and in the variation of the band-edge profile in the continuity and density gradient equations. The impact through other quantities, such as tunneling rates, is currently neglected.

To use geometric fluctuations, in the global `Math` section for a device, specify one or more surfaces. Each surface has a name and is the union of an arbitrary number of interfaces and electrodes. The following example specifies a surface `S2` that consists of all the poly–oxide interfaces in the device and the `channel/gateoxide` region interface:

```
Device "MOS" {
    Math {
        Surface "S2" (
            RegionInterface="channel/gateoxide"
            MaterialInterface="PolySi/Oxide"
        )
        ...
    }
}
```

The order in which regions or materials in the specification of interfaces are named determines the sense of interface displacements: The positive direction points from the region or material named first into the region or material named second.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

The surfaces are then used to activate the noise sources. The following example activates geometric fluctuations for surfaces S1 and S2:

```
Physics {
    Noise(
        GeometricFluctuations "S1"
        GeometricFluctuations "S2"
    ) ...
} ...
}
```

Points on the same surface are correlated according to [Equation 795](#). Points on different surfaces are uncorrelated. The contributions of different surfaces to the noise spectral densities add.

The parameters  $a_{\text{iso}}$ ,  $\hat{a}$ , and  $\lambda$  are specified as the parameters `Amplitude_Iso`, `Amplitude`, and `lambda` in a surface-specific `GeometricFluctuations` parameter set. All three parameters are given in micrometers, and default to zero.  $\lambda$  must be set to positive value.

For example:

```
GeometricFluctuations "S1" {
    lambda = 5e-3
    Amplitude = (9e-4, 3e-4, 0)
    Amplitude_Iso = 1e-4
}
```

#### Note:

For 3D structures, the mesh spacing on the fluctuating interfaces must be small compared to the correlation length  $\lambda$  to be able to resolve the Gaussian in [Equation 795](#). For 2D structures, the mesh does not need to resolve the Gaussian.

The `GeometricFluctuations` keyword supports options, which are specified as follows:

```
GeometricFluctuations "S1" (
    Options = <0..1>                      * default 0
    WeightQuantumPotential = <float>          * default 0.5
    WeightDielectric = <float>                 * default 0
)
```

When you specify geometric variations for a semiconductor–insulator interface, you can specify whether the variation should be applied to both sides of the interface (`Options=0`) or only to the semiconductor side (`Options=1`).

For example, in a MOSFET, if you want to change the thickness of the semiconductor body, but not the thickness of the gate insulator, you can use `Options=0` and include in the `Surface` specification both the (bottom) semiconductor–gate insulator interface and the (top) gate insulator–contact or poly interface, such that the movements of the top and

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

bottom gate insulator interfaces leave the gate insulator thickness unaltered. Alternatively, you can use `option=1` and include in the surface specification only the (bottom) semiconductor–gate insulator interface. Physically, both options are nearly equivalent; however, the latter can sometimes be numerically more accurate.

The values given with `WeightQuantumPotential` and `WeightDielectric` interpolate between two different approximations for the variation terms that arise from the density-gradient quantum correction and the dielectric constant. Calibrating these two parameters can improve the accuracy of geometric variation results. Doing so is usually only worthwhile when computing insulator position variations, while keeping the insulator thickness fixed.

---

## Random Trap Concentration Fluctuations

Trap concentration variations are activated by the `TrapConcentration` option to `Noise`. For each trap level, Sentaurus Device uses a noise source of the form:

$$K_{ii}^{\text{trapconc}}(\vec{r}_1, \vec{r}_2, \omega) = N_i(\vec{r}_1) \frac{\Theta(0.5\text{Hz} - |\nu|)}{1\text{Hz}} \delta(\vec{r}_1 - \vec{r}_2) \quad (796)$$

Here,  $N_i$  is the concentration for the trap level  $i$ .

All trap levels are assumed to be mutually independent. Their contributions to the noise spectral densities add.

The trap concentration fluctuation model accounts for the impact of the trap concentration on the space charge in the Poisson equation and on the trap-related generation–recombination rates in the continuity equations.

---

## Random Workfunction Fluctuations

The random workfunction fluctuations describe the impact of the fluctuation of the workfunction at contacts on insulators and at metal–insulator interfaces. The noise source reads:

$$K^{\text{workfunction}}(\vec{r}_1, \vec{r}_2, \nu) = \frac{\Theta(0.5\text{Hz} - |\nu|)}{1\text{Hz}} a(\vec{r}_1) a(\vec{r}_2) \exp\left[-\frac{(\vec{r}_1 - \vec{r}_2)^2}{\lambda^2}\right] \quad (797)$$

where:

- $\nu = \omega/2\pi$  is the frequency.
- $a$  is the workfunction standard deviation.
- $\lambda$  is the correlation length.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

#### Note:

The spatial correlation in [Equation 797](#) differs from the one created by the randomization procedure used for the sIFM-based approach (see [Workfunction Variations on page 803](#)).

Workfunction fluctuations are activated by the `WorkfunctionFluctuations` option to `Noise`. This option must be followed by a string that denotes the name of a surface that consists exclusively of contacts on insulators and metal–insulator interfaces. For example:

```
Device "MOS" {
    Math {
        Surface "S3" (
            Electrode = "gate"
            MaterialInterface = "Nitride/Aluminum"
        )
    }
    Physics {
        Noise(WorkfunctionFluctuations "S3") ...
    } ...
}
```

The definition of surfaces is described in [Random Geometric Fluctuations on page 790](#). Any number of `WorkfunctionFluctuations` (with different names) can be specified. They are considered to be uncorrelated, and their contributions to the noise spectral densities add.

The parameters  $a$  and  $\lambda$  are specified by `Amplitude` (in eV) and `lambda` (in  $\mu\text{m}$ ) in a surface-specific `WorkfunctionFluctuations` parameter set. For example:

```
WorkfunctionFluctuations "S3" {
    lambda = 0.01
    Amplitude = 0.4
}
```

---

## Random Band Edge Fluctuations

Random band edge fluctuations describe the effect of the variation of the electron affinity  $\delta\chi$  and the band gap  $\delta E_{g,\text{tot}} = \alpha\delta\chi + \delta E_g$ , where  $\alpha$  is a user-specified parameter, and  $\delta\chi$  and  $\delta E_g$  are statistically independent ( $\delta\chi(\vec{r}_1)\delta E_g(\vec{r}_2) = 0$ ) fields with Gaussian correlations. Therefore, noise sources for conduction band and valence band variations are:

$$K^{E_C}(\vec{r}_1, \vec{r}_2, v) = \frac{\Theta(0.5\text{Hz} - |v|)}{1\text{Hz}} a_\chi^2 \exp\left[-\frac{(\vec{r}_1 - \vec{r}_2)^2}{\lambda^2}\right] \quad (798)$$

$$K^{E_V}(\vec{r}_1, \vec{r}_2, v) = \frac{\Theta(0.5\text{Hz} - |v|)}{1\text{Hz}} [(1 + \alpha)^2 a_\chi^2 + a_{E_g}^2] \exp\left[-\frac{(\vec{r}_1 - \vec{r}_2)^2}{\lambda^2}\right] \quad (799)$$

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

For  $a_{E_g} = 0$ , some interesting limiting cases exist:

- For  $\alpha = 0$ , the conduction band and valence band vary in the same direction with the same amplitude.
- For  $\alpha = -1$ , the valence band does not vary at all.
- For  $\alpha = -2$ , the conduction band and the valence band vary in opposite directions.

Band edge fluctuations are activated by the option `BandEdgeFluctuations to Noise`:

```
Physics { Noise(BandEdgeFluctuations <string>) }
```

Here, the string identifies a volume specification that determines where, in the device, band edge fluctuations are active. Named volumes are specified in the device global `Math` section by providing lists of regions and materials that belong to the volume:

```
Math {
    Volume <string> (
        Regions = (<string>...)
        Materials = (<string>...)
    )
}
```

The dimensionless parameter  $\alpha$ , the correlation length  $\lambda$  ( $\mu\text{m}$ ), and the amplitudes  $a_\chi$  and  $a_{E_g}$  are specified as `Chi2Eg`, `Lambda`, `Amplitude_Chi`, and `Amplitude_Eg` in a named global parameter set `BandEdgeFluctuations`:

```
BandEdgeFluctuations <string> {
    Amplitude_Chi = <float>           * in eV, default 0
    Amplitude_Eg = <float>             * in eV, default 0
    Chi2Eg = <float>                  * dimensionless, default 0
    Lambda = <float>                  * in  $\mu\text{m}$ , no default (required)
}
```

The string is the name with which the band edge fluctuations are activated as a `Noise` option in the device global `Physics` section.

---

## Random Metal Conductivity Fluctuations

Metal conductivity fluctuations are activated by the option `ConductivityFluctuations to Noise`:

```
Physics { Noise(ConductivityFluctuations <string>) }
```

Here, `<string>` identifies a named volume specification that determines where, in the device, metal conductivity fluctuations are active. For the specification of named volumes, see [Random Band Edge Fluctuations on page 794](#).

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Sources

Metal conductivity fluctuations are assumed to have Gaussian spatial correlations. You specify parameters for metal conductivity fluctuations in the named global `ConductivityFluctuations` parameter set in the parameter file:

```
ConductivityFluctuations <string> {
    Amplitude = <float>           * amplitude in A/cmV
    Lambda= <float>                * correlation length in um
}
```

The string is the name with which the metal conductivity fluctuations are activated as a `Noise` option in the device global `Physics` section.

---

## Random Dielectric Constant Fluctuations

Random dielectric constant fluctuations are activated by the option `EpsilonFluctuations` to `Noise`:

```
Physics {
    Noise(EpsilonFluctuations <string>)
}
```

Here, `<string>` identifies a named volume specification that determines where, in the device, dielectric constant fluctuations are active. For the specification of named volumes, see [Random Band Edge Fluctuations on page 794](#). Random dielectric constant fluctuations are assumed to have Gaussian spatial correlations.

Parameters for random dielectric constant fluctuations are specified in the named global `EpsilonFluctuations` parameter set in the parameter file:

```
EpsilonFluctuations <string> {
    Amplitude = <float>           * amplitude, dimensionless
    Lambda= <float>                * correlation length in um
}
```

The string is the name with which the dielectric constant fluctuations are activated as a `Noise` option in the device global `Physics` section.

---

## Noise From SPICE Circuit Elements

To take into account the noise generated by SPICE circuit elements, specify the `CircuitNoise` option to `ACCoupled` (see *Compact Models User Guide*, Chapter 1). The form of the noise source for a particular circuit element is defined by the respective compact model.

Due to a restriction of the SPICE noise models, SPICE circuit elements contribute only to the autocorrelation noise. For cross-correlation noise, Sentaurus Device considers SPICE

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

circuit elements as noiseless. Non-SPICE compact circuit elements do not implement noise at all and, therefore, Sentaurus Device always treats them as noiseless.

---

## Statistical Impedance Field Method

The statistical impedance field method (sIFM) can be applied to doping concentration, trap concentration, semiconductor–insulator and insulator–insulator interface positions, and the workfunction at contacts on insulators and metal–insulator interfaces. It creates a large number of randomized variations of the quantities under investigation (dopant concentrations, trap concentrations, or workfunction) and computes the modification of the device characteristics in linear response.

The sIFM is specified in the global `Math` section:

```
Math {  
    RandomizedVariation <string> (  
        NumberOfSamples = <int>  
        Randomize = <int>  
        ExtrudeTo3D  
        RandomField(...)  
    )  
    ...  
}
```

Any number of `RandomizedVariation` specifications can be present, each of which specifies the sample size and the random seed for one set of variations. The sets are distinguished by the optional string after the `RandomizedVariation` keyword.

`NumberOfSamples` determines the number of variations in a set. `Randomize` determines the seed for the random number generator. For a value of zero (the default), the seed value is selected automatically and differently in each simulation run. A negative value disables the sIFM, and a positive value is directly taken as the seed for the random number generator. The last possibility allows to reproduce results for different simulation runs of the same device, on the same platform, and the same release of Sentaurus Device.

For 2D structures, the keyword `ExtrudeTo3D` instructs Sentaurus Device to perform randomization for a 3D structure that is obtained by extruding the 2D structure to a width given by the `AreaFactor`. This is necessary to correctly model variations with spatial correlations, but it will increase runtime for these variations. By default, randomization is performed in two dimensions, which corresponds to variations that are perfectly correlated in the third spatial direction. `ExtrudeTo3D` has no effect on 3D structures.

`RandomField` allows you to specify the statistical properties of the spatial correlations for this `RandomizedVariation` specification (see [Spatial Correlations and Random Fields on page 799](#)).

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

The quantities to be randomized are specified in a device-global `Physics` section:

```
Device <string> {
    Physics {
        RandomizedVariation <string> (<specifications>)
    }
}
```

The string after `RandomizedVariation` in the `Physics` section is matched with the one after `RandomizedVariation` in the global `Math` section, and `<specifications>` is described along with the individual variations in the following sections.

For one `ACCoupled` statement, for each `RandomizedVariation` specification, the sIFM creates two files per `ObservationNode`: one for voltage and one for current variation at the node. The file names are derived from the base name given by `ACEExtract`, the string after the `RandomizedVariation` keyword, the name of the type of variation (current or voltage), the node name, and the extension `.csv`. The file contains comma separated values (CSV), a very simple format that can be imported by many applications (for example, spreadsheet programs).

The first line contains the column headings, using double-quoted strings. Each subsequent line corresponds to one bias point and one frequency.

The first couple of columns contain the data specified in the `ACPlot` statement. They serve to connect each line to a bias point. The rest of the columns within each line correspond to the current or voltage shift for one particular randomized profile. For all lines, the same column always corresponds to the same profile. For frequencies larger than 0.5 Hz, the shifts are zero.

---

## Options Common to sIFM Variations

All variations supported by sIFM provide the keywords `SpatialShape`, `SpaceMid`, and `SpaceSig`. They allow you to multiply the variation by a space-dependent factor, and they work in the same way as for traps (see [Energetic and Spatial Distribution of Traps on page 544](#)). The shape functions must adequately be resolved by the mesh. The following example activates random doping variations, which are damped with a Gaussian function on a length scale of 10 nm around the origin:

```
Physics {
    RandomizedVariation "rnd" (
        Doping(
            SpatialShape = Gaussian
            SpaceMid = (0 0 0)
            SpaceSig = (0.01 0.01 0.01)
        )
    )
}
```

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

Using `SpatialShape=Uniform` instead of `SpatialShape=Gaussian` would switch off the doping variation completely for points outside a cube of 20nm side length centered at the origin.

By default, `SpatialShape` is `Uniform`, and `SpaceSig` is a vector with very large components. Therefore, by default, the space-dependent factor is one everywhere. If the mesh has dimension  $d$ , only the  $d$  first components of `SpaceMid` and `SpaceSig` are significant; additional components are ignored.

---

## Spatial Correlations and Random Fields

Several variations supported by sIFM are spatially correlated. You can specify the statistical properties of the correlation either directly in the specification of the variation or by declaring `RandomField` in the `RandomizedVariation` section in the global `Math` section. If the statistical properties are specified directly in the specification of the variation, the variation is statistically independent from all other variations. In contrast, if `RandomField` is declared, all variations in the same `RandomizedVariation` section are correlated, and direct specifications are ignored.

Random fields are declared like this:

```
Math {
    RandomizedVariation <string> (
        RandomField(
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector> * in μm
            Lambda = <vector> * in μm
            Resolution = <vector>
            MaxInternalPoints = <int>
        )
        ...
    )
}
```

The spatial domain covered by a random field is obtained automatically from the spatial domains in which the variations that use it are active. A random field can have more than one component.

The number of components in the random field is the largest number required by any of the variations that refer to the random field. Only band edge variations require two components; the other variations require only one component. The components are numbered starting from zero. Variations with only one component always use the zero-th component of the random field.

`CorrelationFunction` determines the way in which spatial correlations are modeled.

If `CorrelationFunction=Grain` (default), first, grains are generated randomly. The creation of grains works as described in [Metal Workfunction Randomization on page 299](#). If

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

AverageGrainSize is the vector  $(\lambda_x, \lambda_y, \lambda_z)$ , the vertex at coordinate  $(x, y, z)$  belongs to the grain with the center  $(x_0, y_0, z_0)$  for which  $(x - x_0)^2/\lambda_x^2 + (y - y_0)^2/\lambda_y^2 + (z - z_0)^2/\lambda_z^2$  becomes smallest.

For each grain and for each component of the random field, a random number  $g$  in the range from 0 to 1 is selected, with uniform probability and statistically independent from the random number for other grains or other components. For variations that use a list of discrete values for the variation,  $g$  is mapped into this list according to the variation-specific probability for the values in the list. If  $g$  is close to zero, the first value in the list will be taken. If  $g$  is close to 1, the last value in the list will be taken. For variations with continuous values,  $g$  will be mapped by a monotonically increasing function to the probability distribution for the variation.

If CorrelationFunction is Exponential or Gaussian, a Fourier approach is used to create fields  $g(\vec{r})$  of exponentially correlated or Gaussian correlated, dimensionless random numbers. The fields for different components of the random field are uncorrelated.

The correlation length is given (in  $\mu\text{m}$ ) by the components  $(\lambda_x, \lambda_y, \lambda_z)$  of Lambda:

$$g(x, y, z)g(x_0, y_0, z_0) = \exp(-\sqrt{(x - x_0)^2/\lambda_x^2 + (y - y_0)^2/\lambda_y^2 + (z - z_0)^2/\lambda_z^2}) \quad (800)$$

$$g(x, y, z)g(x_0, y_0, z_0) = \exp(-(x - x_0)^2/\lambda_x^2 - (y - y_0)^2/\lambda_y^2 - (z - z_0)^2/\lambda_z^2) \quad (801)$$

The physical variation is obtained by multiplying  $g(\vec{r})$  by a variation-specific amplitude.

For Gaussian and exponential correlations, Resolution (one to three dimensionless positive components; default (0.25 0.25 0.25)) can be used to specify how accurately spatial correlations should be resolved, for each spatial axis. The components specify this accuracy in units of Lambda. One possible application for Resolution is to use a large value for one axis to pretend that the domain is nearly flat along this direction, which reduces the dimension of the Fourier transform and can give a substantial speedup.

For Gaussian and exponential correlations, MaxInternalPoints specifies the maximum number of internal points used to compute the randomization. The default value is a very large integer. If the number of internal points used to compute the randomization exceeds the value set with MaxInternalPoints, Sentaurus Device terminates. The purpose of this is to avoid excessive computation time, which scales superlinearly with the number of internal points.

To reduce the number of internal points, restrict the randomized domain, or increase Lambda or Resolution.

You can specify Lambda and AverageGrainSize as vectors with less than three components. In this case, the missing components are assumed equal to the last given component.

All options that are available in the RandomField section are also available as options to the specification of all variations with spatial correlation, with the same meaning of keywords.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

The specification of variations with spatial correlation supports a string, which allows you to put several such variations into the same RandomizedVariation section. Their contribution is added. For example:

```
Physics {
    RandomizedVariation "rv" (
        Geometric "g1" ( ... )
        Geometric "g2" ( ... )
    )
}
```

---

## Doping Variations

Doping variations are enabled using the Doping keyword:

```
Physics {
    RandomizedVariation <string> (
        Doping (
            Type = Doping | Acceptor | Donor
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
            -Mobility
            -BandgapNarrowing
        )
    )
}
```

For RandomizedVariation specifications in the Math section for which no RandomizedVariation specification of the same name is present in the Physics section, dopant variations are enabled automatically.

Type determines which dopants to randomize. For Acceptor, only acceptors are randomized. For Donor, only donors are randomized. For Doping (default), both are randomized.

For the options of Doping, see [Options Common to sIFM Variations on page 798](#).

Acceptor and donor concentrations are randomized independently, assuming dopants are spatially uncorrelated and using a Poisson distribution function.

This means that for a vertex  $i$  of the mesh with a box volume  $V_i$  and an average doping concentration  $N_i$ , the probability to find exactly  $k$  dopants in the box for vertex  $i$  is:

$$P_i(k) = \frac{(N_i V_i)^k}{k!} \exp(-N_i V_i) \quad (802)$$

The volume  $V_i$  is a 3D volume. If the simulated structure is 2D, the AreaFactor is taken into account to compute  $V_i$ . [Equation 802](#) is consistent with the assumptions that are used to obtain [Equation 793](#).

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

The sIFM accounts for the impact of dopant concentration on space charge, mobility, and bandgap narrowing. The options `-Mobility` and `-BandgapNarrowing` disable the impact due to mobility and bandgap narrowing, respectively.

#### Note:

It is strongly advisable to use either both the `-BandgapNarrowing` and `-Mobility` options or neither option. Otherwise, inconsistencies can arise when the mobility depends indirectly on bandgap narrowing. Even when this dependency is weak, the inconsistency can lead to large errors in IFM results.

When the keyword `RandomizedDoping` appears in a device-specific `Plot` section, all randomized acceptor and donor profiles are written to the `Plot` file. The individual profiles are identified by a number in their name. This number is the same one that appears in the first line of the `.csv` files.

#### Note:

If the number of random samples is large, plotting randomized doping profiles can occupy a large amount of memory.

---

## Trap Concentration Variations

For the sIFM, variations of the random trap concentration are activated with the keyword `TrapConcentration`:

```
Physics {
    RandomizedVariation <string> (
        TrapConcentration (
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
        )
    )
}
```

All trap levels are randomized independently, assuming traps are spatially uncorrelated. The same expression as for doping concentration, [Equation 802](#), is used. For the options of `TrapConcentration`, see [Options Common to sIFM Variations on page 798](#).

The sIFM accounts for the impact of the trap concentration on the space charge in the Poisson equation and on the trap-related generation–recombination rates in the continuity equations.

## Workfunction Variations

For the sIFM, workfunction variations are activated and controlled using the `Workfunction` keyword:

```
Physics {
    RandomizedVariation <string> (
        Workfunction <string> (
            Surface = <string>
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector>
            Lambda = <vector>
            Resolution = <vector>
            MaxInternalPoints = <int>
            GrainProbability = (<float>...)
            GrainWorkfunction = (<float>...) * in eV
            Amplitude = <float> * in eV
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
        )
    )
}
```

`Surface` specifies the name of a surface that consists of contacts on insulators and metal–insulator interfaces only. The specification of surfaces is described in [Random Geometric Fluctuations on page 790](#). The workfunction is randomized for interfaces and contacts that are part of the given `Surface`.

`GrainWorkfunction` is a list of workfunction values (in eV) that can occur for a metal grain. `GrainProbability` is a list that gives the probabilities with which these workfunctions occur. These two parameters are used for `Grain` correlations. `Amplitude` (in eV) is a multiplier to the dimensionless field  $g(\vec{r})$  that is used for Gaussian and exponential correlations. For details, see [Spatial Correlations and Random Fields on page 799](#).

As the sIFM models the effect of deviations, the values of `GrainWorkfunction` are adjusted to have an average that agrees with the nominal workfunction specified for the electrode or metal. Therefore, adding the same value to all `GrainWorkfunction` values will not affect the results.

For the other options of `Workfunction`, see [Options Common to sIFM Variations on page 798](#) and [Spatial Correlations and Random Fields on page 799](#).

The sIFM accounts for the impact of the workfunction on the boundary condition for the electrostatic potential at metal–insulator interfaces and at contacts on insulators.

## Geometric Variations

For the sIFM, geometric variations are activated and controlled using the `Geometric` keyword:

```
Physics {
    RandomizedVariation <string> (
        Geometric <string> (
            Surface = <string>
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector>
            Lambda = <vector>
            Resolution = <vector>
            MaxInternalPoints = <int>
            Amplitude = <vector>           * in micrometer
            Amplitude_Iso = <float>        * in micrometer
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
            Options = <0..1>
            WeightQuantumPotential = <float>
            WeightDielectric = <float>
        )
    )
}
```

`Surface` specifies the name of a surface that consists of electrodes on insulator, metal–insulator, semiconductor–insulator, and insulator–insulator interfaces only. The specification of surfaces is described in [Random Geometric Fluctuations on page 790](#). The interface position is randomized for interfaces that are part of the given `Surface`.

For `Grain` correlations, for each grain, a random number  $g$  is selected from a Gaussian distribution of average zero and variance one. For each surface point that is contained in the grain, the interface shift along the normal direction  $n$  is given by  $gn \cdot (a_{\text{iso}} + \hat{a} \cdot n)$ . Here,  $a_{\text{iso}}$  and  $\hat{a}$  are the values specified (in  $\mu\text{m}$ ) by `Amplitude_Iso` and `Amplitude`. For Gaussian and exponential variation, the field  $g(\vec{r})$  of dimensionless correlated random numbers determines the interface shift along the normal direction  $n$  in a point  $\vec{r}$  on the surface as  $g(\vec{r})n \cdot (a_{\text{iso}} + \hat{a} \cdot n)$ . As above,  $a_{\text{iso}}$  and  $\hat{a}$  are the values specified by `Amplitude_Iso` and `Amplitude`. For more details, see [Spatial Correlations and Random Fields on page 799](#).

The keywords `Options`, `WeightQuantumPotential`, and `WeightDielectric` work as explained in [Random Geometric Fluctuations on page 790](#).

For the other options of `Geometric`, see [Options Common to sIFM Variations on page 798](#) and [Spatial Correlations and Random Fields on page 799](#).

## Band Edge Variations

For the sIFM, band edge variations are activated and controlled using the `BandEdge` keyword:

```
Physics {
    RandomizedVariation <string> (
        BandEdge <string> (
            Volume = <string> * required
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector>
            Lambda = <vector>
            Resolution = <vector>
            MaxInternalPoints = <int>
            ChiComponent = <0..1> * default 0, RandomField
            * component to use
            EgComponent = <0..1> * default 1, RandomField
            * component to use
            GrainProbability = (<float>...) * dimensionless, nonnegative
            GrainChi = (<float>...) * ev, same size as
            * GrainProbability
            GrainEg = (<float>...) * ev, default 0, same size as
            * or smaller than
            * GrainProbability
            Chi2Eg = <float> * dimensionless, default 0
            Amplitude_Chia = <float> * ev
            Amplitude_Eg = <float> * ev
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
        )
    )
}
```

`Volume` specifies a named volume that determines where band edge variations are active. For the specification of named volumes, see [Random Band Edge Fluctuations on page 794](#).

For `Grain` correlations, `GrainProbability` is a list that gives the probabilities with which these shifts occur. `GrainChi` is a list of affinity shifts  $\delta\chi$  (in eV) with the same number of values as `GrainProbability`. Bandgap variations are computed as  $\delta E_{g,tot} = \alpha\delta\chi + \delta E_g$ , where  $\alpha$  is given by the `Chi2Eg` keyword, and  $\delta E_g$  are values selected (statistically independently from the  $\delta\chi$ , but using the same `GrainProbability` list) from the list `GrainEg` (in eV). `GrainEg` has, at most, as many values as `GrainProbability`. Missing values in `GrainEg` are assumed to be zero.

The values in `GrainChi` and `GrainEg` automatically adjust to have zero average. Therefore, adding a constant to all components of these lists has no effect.

For exponential and Gaussian correlations,  $\delta\chi$  and  $\delta E_g$  are obtained by multiplying `Amplitude_Chia` and `Amplitude_Eg` (in eV) by the zero-th and first component of the dimensionless field  $g(\vec{r})$  that is used for Gaussian and exponential correlations.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Statistical Impedance Field Method

For all correlations, if `RandomField` is used, `ChiComponent` and `EgComponent` allow you to specify the component of the random field used to determine  $\delta\chi$  and  $\delta E_g$ .

For details about correlations, see [Spatial Correlations and Random Fields on page 799](#).  
For the other remaining options, see [Options Common to sIFM Variations on page 798](#).

---

## Metal Conductivity Variations

For the sIFM, metal conductivity variations are activated and controlled using the `Conductivity` keyword:

```
Physics {
    RandomizedVariation <string> (
        Conductivity<string> (
            Volume = <string>                                * required
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector>
            Lambda = <vector>
            Resolution = <vector>
            MaxInternalPoints = <int>
            GrainProbability = (<float>...)      * dimensionless, nonnegative
            GrainConductivity = (<float>...)      * A/cmV, size as
                                                * GrainProbability
            Amplitude = <float>                      * A/cmV
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
        )
    )
}
```

`Volume` specifies a named volume that determines where metal conductivity variations are active. For the specification of named volumes, see [Random Band Edge Fluctuations on page 794](#).

For `Grain` correlations, `GrainProbability` is a list that gives the probabilities with which these shifts occur. `GrainConductivity` is a list of a conductivity shifts (in A/cmV) with the same number of values as `GrainProbability`. The values in `GrainConductivity` are adjusted to have zero average. Therefore, adding a constant to all components of this list has no effect.

`Amplitude` (in A/cmV) is a multiplier to the dimensionless field  $g(\vec{r})$  that is used for Gaussian and exponential correlations.

For details about correlations, see [Spatial Correlations and Random Fields on page 799](#).  
For the other remaining options, see [Options Common to sIFM Variations on page 798](#).

## Dielectric Constant Variations

For the sIFM, relative dielectric constant variations are activated and controlled using the **Epsilon** keyword:

```
Physics {
    RandomizedVariation <string> (
        Epsilon <string> (
            Volume = <string> * required
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector>
            Lambda = <vector>
            Resolution = <vector>
            MaxInternalPoints = <int>
            GrainProbability = (<float>...) * dimensionless, nonnegative
            GrainEpsilon = (<float>...) * dimensionless, size as
                           * GrainProbability
            Amplitude = <float> * dimensionless
            SpatialShape = Uniform | Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
        )
    )
}
```

**Volume** specifies a named volume that determines where relative dielectric constant variations are active. For the specification of named volumes, see [Random Band Edge Fluctuations on page 794](#).

For **Grain** correlations, **GrainProbability** is a list that gives the probabilities with which these shifts occur. **GrainEpsilon** is a list of a relative dielectric constant shifts with the same number of values as **GrainProbability**. The values in **GrainEpsilon** are adjusted to have zero average. Therefore, adding a constant to all components of this list has no effect.

**Amplitude** (dimensionless) is a multiplier to the dimensionless field  $g(\vec{r})$  that is used for Gaussian and exponential correlations.

For details about correlations, see [Spatial Correlations and Random Fields on page 799](#). For the other remaining options, see [Options Common to sIFM Variations on page 798](#).

---

## Doping Profile Variations

Doping profile variations are activated and controlled using the **DopingVariation** keyword:

```
Physics {
    RandomizedVariation <string> (
        DopingVariation <string> (
            CorrelationFunction = Grain | Exponential | Gaussian
            AverageGrainSize = <vector>
        )
    )
}
```

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Deterministic Variations

```
Lambda = <vector>
Resolution = <vector>
MaxInternalPoints = <int>
SFactor = <string>
Amplitude = <vector>      * µm
Amplitude_Iso = <float>    * µm
Conc = <float>             * 1/ccm
Type = Doping | Acceptor | Donor
SpatialShape = Uniform | Gaussian
SpaceMid = <vector>
SpaceSig = <vector>
-Mobility
-BandgapNarrowing
)
)
}
```

The options `SFactor`, `Conc`, `Type`, `Amplitude`, and `Amplitude_Iso` are used to describe a doping profile shift based on the gradient of a dataset, as described in [Deterministic Doping Variations on page 809](#). In addition to what is described there, for sIFM, the shift is multiplied by a spatially correlated, dimensionless, random field.

For grain correlations, the random number in each grain is Gaussian distributed, with zero average and variance of one. For exponential and Gaussian correlations, the dimensionless random field is the one obtained from the Fourier approach.

By default, the impact of doping concentration on space charge, mobility, and bandgap narrowing is accounted for. The options `-Mobility` and `-BandgapNarrowing` disable the impact due to mobility and bandgap narrowing, respectively.

#### Note:

It is strongly advisable to use either both the `-BandgapNarrowing` and `-Mobility` options or neither option. Otherwise, inconsistencies can arise when the mobility depends indirectly on bandgap narrowing. Even when this dependency is weak, the inconsistency can lead to large errors in IFM results.

For the other options of `DopingVariation`, see [Options Common to sIFM Variations on page 798](#) and [Spatial Correlations and Random Fields on page 799](#).

---

## Deterministic Variations

For deterministic variations, you specify the variations directly, by specifying the actual deviation in doping, geometry, or model parameters. Sentaurus Device computes the effect of the variations on the observation node voltages and currents in a linear response. As for random fluctuations, deterministic variations are assumed to be different from zero only for analysis frequencies up to 0.5 Hz.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Deterministic Variations

Compared to random fluctuations, deterministic variations give you more control over the variation and are easier to understand, because no statistical interpretation is required and no second-order moments appear. In the case of geometric variations, the computational effort is smaller as well.

---

## Deterministic Doping Variations

Deterministic doping variations are specified as an option to `DeterministicVariation` in the `Physics` section:

```
Physics {
    DeterministicVariation(
        DopingVariation <name> (
            SFactor = <string>
            Amplitude = <vector>
            Amplitude_Iso = <float>
            Amplitude_Abs = <float>
            Conc = <float>
            Factor = <float>
            Type = Doping | Acceptor | Donor
            SpatialShape = Gaussian | Uniform
            SpaceMid = <vector>
            SpaceSig = <vector>
            -Mobility
            -BandgapNarrowing
        ) ...
    ) ...
}
```

Here, `<name>` is a string that names the variation and will be used to identify output to the `ACEExtract` file. If it coincides with the name of a geometric or a parameter variation, the contributions of the variations are added.

`SFactor` specifies a scalar dataset. Allowed datasets are doping concentrations and `PMIUserFields`. If you do not specify any dataset, a constant density of  $1 \text{ cm}^{-3}$  is assumed.

By specifying `Conc`, the concentration is normalized and then multiplied by the concentration specified with `Conc`. `Conc` is given in  $\text{cm}^{-3}$ . If `Conc` is not specified or is zero, the values from the dataset will be used unaltered.

If the `SFactor` dataset is a `PMIUserField` dataset or another dataset that is not a density, Sentaurus Device cannot properly determine the units of the dataset and, therefore, it interprets its value with a different scaling constant to the one you intended. Specifying `Conc` can be useful in this case, as the normalization of the `SFactor` eliminates the ambiguous units.

By default, the resulting concentration  $X$  determines the fluctuation  $\delta N$  directly,  $\delta N = X$ . If you use `Amplitude_Iso`, `Amplitude`, or `Amplitude_Abs` to specify nonzero isotropic or

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Deterministic Variations

vectorial amplitudes  $a_{\text{iso}}$ ,  $\hat{\vec{a}}$ , and  $\hat{\vec{a}}_{\text{abs}}$ , the fluctuation is computed from  $X$  as  $\delta N = -\hat{\vec{a}} \cdot \nabla X - a_{\text{iso}} |\nabla X| - \hat{\vec{a}}_{\text{abs}} \cdot \text{abs}(\nabla X)$ , where  $\text{abs}(\nabla X)$  is the vector obtained by taking the absolute value of  $\nabla X$  component-wise. The vectorial amplitude models the small displacement of a profile, and the isotropic amplitude models a shift of a doping front (perpendicular to the equi-doping lines). For this to work properly, the mesh must accurately resolve the variations of  $X$ . Both amplitudes are specified in  $\mu\text{m}$ .

Optionally, a dimensionless value `Factor` can be specified, which is multiplied by the resulting  $\delta N$ . The factor defaults to one. Its purpose is to easily specify variations that are a certain percentage of a given doping data field.

Using the keywords `SpatialShape`, `SpaceMid`, and `SpaceSig`, you can multiply  $\delta N$  by either a Gaussian function or a window function. These keywords work in the same way as for traps (see [Energetic and Spatial Distribution of Traps on page 544](#)). The shape functions must adequately be resolved by the mesh.

`Type` selects whether the variation applies to the acceptor concentration ( $\delta N_A = \delta N$ ), the donor concentration ( $\delta N_D = \delta N$ ), or doping as a whole. In the latter case, a negative variation increases the acceptor concentration; a positive variation increases donor concentration.

By default, the impact of doping concentration on space charge, mobility, and bandgap narrowing is accounted for. The options `-Mobility` and `-BandgapNarrowing` disable the impact due to mobility and bandgap narrowing, respectively.

#### Note:

It is strongly advisable to use either both the `-BandgapNarrowing` and `-Mobility` options or neither option. Otherwise, inconsistencies can arise when the mobility depends indirectly on bandgap narrowing. Even when this dependency is weak, the inconsistency can lead to large errors in IFM results.

For a summary of options to `DopingVariation`, see [Table 293 on page 1673](#).

---

## Deterministic Geometric Variations

Deterministic geometric variations are specified as an option to `DeterministicVariation` in the `Physics` section:

```
Physics {
    DeterministicVariation(
        GeometricVariation <name> (
            Surface = <string>
            Amplitude = <vector>
            Amplitude_Iso = <float>
            SpatialShape = Gaussian
            SpaceMid = <vector>
            SpaceSig = <vector>
```

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Deterministic Variations

```
    Options = <0..1>
    WeightQuantumPotential = <float>
    WeightDielectric = <float>
)
)
}
}
```

Here, `<name>` is a string that names the variation and will be used to identify output to the `ACEExtract` file. If it coincides with the name of a doping or a parameter variation, the contributions of the variations are added.

`Surface` identifies the displaced interfaces. Its specification is described in [Random Geometric Fluctuations on page 790](#). The amount of displacement along the positive normal in a point  $r$  on the surface is determined by `Amplitude_Iso` ( $a_{iso}$ ) and `Amplitude` ( $a$ ), both specified in  $\mu\text{m}$ , as  $\delta s(\vec{r}) = a_{iso}(\vec{r}) + \vec{a}(\vec{r}) \cdot \vec{n}(\vec{r})$ .

Using the keywords `SpatialShape`, `SpaceMid`, and `SpaceSig`, you can multiply  $\delta s$  by a Gaussian function.

These keywords work in the same way as for traps (see [Energetic and Spatial Distribution of Traps on page 544](#)). The Gaussian must be resolved adequately by the mesh.

The keywords `Options`, `WeightQuantumPotential`, and `WeightDielectric` work as explained in [Random Geometric Fluctuations on page 790](#).

For a summary of options of `GeometricVariation`, see [Table 307 on page 1681](#).

---

## Parameter Variations

Sentaurus Device allows to compute the linear response to the variation of any parameter that can be ramped. The parameter variations are specified as an option to `DeterministicVariation` in the `Physics` section:

```
Physics {
    DeterministicVariation (
        ParameterVariation <name> (
            (
                Material = <string>
                Region = <string>
                MaterialInterface = <string>
                RegionInterface = <string>
                Model = <string>
                Parameter = <string>
                Value = <float>
                Factor = <float>
                Summand = <float>
            )...
        )...
    )...
}
```

## Chapter 23: Noise, Fluctuations, and Sensitivity

### IFM Section

Here, <name> is a string that names the variation and is used to identify output to the ACEExtract file. If it coincides with the name of a doping or a geometry variation, the contributions of the variations are added.

The option of ParameterVariation is a list of an arbitrary number of individual parameter specifications, each of which is enclosed by a pair of parentheses. All these variations are performed together to compute their cumulative impact.

Material, Region, MaterialInterface, or RegionInterface specify the location of the parameter that is varied. At most, one of these keywords must be specified. Model specifies the name of the model to which the varied parameter belongs, and Parameter is the name of the parameter within this model. All these keywords specify a parameter in the same way as for parameter ramping (see [Ramping Physical Parameter Values on page 126](#)).

To specify the correct location for a parameter can be complicated. [Combining Parameter Specifications on page 78](#) explains how the specifications in the parameter file determine the location from which the parameters used in the computation are taken.

Each varied parameter has an original value  $\gamma$  (the default value, or the value that was set in the parameter file).

The modified value  $\gamma'$  can be given by one of two ways:

- Directly by specification of the modified value using `value= $\gamma'$`
- By using the keywords Factor and Summand:  $\gamma' = \text{Factor} \cdot \gamma + \text{Summand}$

Sentaurus Device assembles the right-hand side twice: once with the original parameters, and once with the modified parameters. Then, the difference in the right-hand sides is used to compute the variation of output characteristics in the linear response. However, as the right-hand sides are not linear in the parameters, the method is not perfectly linear, which becomes visible if the parameter variation  $\gamma' - \gamma$  is not small. On the other hand, if the parameter variation becomes too small, numeric noise can obscure the results.

---

## IFM Section

Instead of defining the noise and variation models in the Math and Physics sections, you can also define all these models in the IFM section as follows:

```
IFM {
    Noise ( ... )
    RandomizedVariation ( ... )
    DeterministicVariation ( ... )
    ...
}
```

## Chapter 23: Noise, Fluctuations, and Sensitivity

### IFM Section

Parameters of the `Noise` model can be defined in the `IFM` section as well:

```
IFM {
    Noise (
        MonopolarGRNoise ( e_alpha = 1 ... )
        ...
    )
    ...
}
```

[Table 141](#) lists all the `Noise` model parameters.

*Table 141 Noise model parameters*

Model	Symbol	Default	Unit
MonopolarGRNoise	e_alpha	1	1
	h_alpha	1	1
	e_tau	1e-7	s
	h_tau	1e-7	s
FlickerGRNoise	e_alpha_H	2e-3	1
	h_alpha_H	2e-3	1
	e_tau0	1e-6	s
	h_tau0	1e-6	s
	e_tau1	3e-4	s
	h_tau1	3e-4	s
GeometricFluctuations	lambda	0	μm
	Amplitude_Iso	-3e303	μm
	Amplitude	(-3e303, -3e303, -3e303)	μm
WorkfunctionFluctuations	lambda	0	μm
	Amplitude	-1	eV
BandEdgeFluctuations	lambda	0	μm
	Amplitude_chi	0	eV

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Impedance Field Method

Table 141 Noise model parameters (Continued)

Model	Symbol	Default	Unit
EpsilonFluctuations	Amplitude_Eg	0	eV
	Chi2Eg	0	1
ConductivityFluctuations	lambda	0	μm
	Amplitude	-1	A cm <sup>-1</sup> V <sup>-1</sup>
EpsilonFluctuations	lambda	0	μm
	Amplitude	-1	1

## Impedance Field Method

The impedance field method splits noise and fluctuation analysis into two tasks. The first task is to provide models for local microscopic fluctuations inside the devices. The selection of the appropriate models depends on the problem. You have to select the models according to the kind of fluctuation that interests you. The second task is to determine the impact of the local fluctuations on the terminal characteristics. To solve this task, the response of the contact voltage or contact current to local fluctuation is assumed to be linear. For each contact, Green's functions are computed that describe this linear relationship. In contrast to the first task, the second task is purely numeric, as the Green's functions are completely specified by the transport model.

A Green's function describes the response  $G_\xi^c(x, \omega)$  of the current or voltage at node  $c$  due to a perturbation of quantity  $\xi$  at location  $x$  with angular frequency  $\omega$ . Particularly important is the case where  $\xi$  is the right-hand side of the partial differential equation for a solution variable  $\phi$  (see [Equation 37](#)),  $n$  or  $p$  (see [Equation 57](#)),  $T_n$  (see [Equation 76](#)),  $T_p$  (see [Equation 77](#)), or  $T$ .

Given the Green's function and a variation  $\delta\xi$  of the quantity  $\xi$ , Sentaurus Device can compute the current response at node  $c$  as:

$$\delta I_c = G_\xi^c(x, \omega) \delta\xi(x, \omega) dx \quad (803)$$

Here, the integral runs over the entire device. For the sIFM (see [Statistical Impedance Field Method on page 797](#)),  $\delta\xi$  is obtained by the randomization procedure specific to the type of variation and, for deterministic variations (see [Deterministic Variations on page 808](#)),  $\delta\xi$  is given by users.

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Output Data

For noise (and noise-like descriptions of static variations; see [Noise Sources on page 787](#)), it is assumed that the expectation value  $\delta\xi$  of  $\delta\xi$  vanishes, and the second-order statistic moment, the so-called noise source, is considered:

$$K(x, x', \omega) = \delta\xi(x, \omega) \cdot \delta\xi^*(x', \omega) \quad (804)$$

From the noise source, the noise current spectral densities (or in the static case, the variances and covariances) are obtained as:

$$S_I^{c_1, c_2} = G_\xi^{c_1}(x, \omega) K(x, x', \omega) G_\xi^{c_2*}(x', \omega) dx dx' \quad (805)$$

Similar relations are used for the noise voltage spectral densities  $S_V$ .

Some of the noise sources are local,  $K(x, x', \omega) = \delta(x - x')K(x, \omega)$ , which allows you to reduce the number of integrations in [Equation 805](#) to one.  $K(x, \omega)$  is called the *local noise source* (LNS), and the integrand of [Equation 805](#) in this case is called the *local noise current spectral density* (LNISD), and the corresponding integrand for  $S_V$  is called the *local noise voltage spectral density* (LNVSD).

---

## Noise Output Data

Several variables can be plotted during noise analysis. For each device, a `NoisePlot` section can be specified similar to the `Plot` section, where the data to be plotted is listed. Besides the standard data, additional noise-specific data or groups of data can be specified, as listed in [Table 142](#). In the tables, the abbreviations LNS (local noise source) and LNVSD (local noise voltage spectral density) are used.

Autocorrelation data refers to [Equation 805](#) when  $c_1 = c_2$ . Data selected in the `NoisePlot` section is plotted for each device and observation node at a given frequency into a separate file. File names with the following format are used:

```
<noise-plot>_<device-name>_<ob-node>_<number>_acgf_des.tdr
```

where `<noise-plot>` is the prefix specified by the `NoisePlot` option to `ACCCoupled`.

### Note:

Only the noise data specified in a `Noise` section without a name, or with an empty string as a name, will be plotted.

In the case of  $c_1 \neq c_2$  in [Equation 805](#), node cross-correlation spectra are computed and integrands become complex. Data specified in the `NoisePlot` section is plotted for each device and each pair of observation nodes at a given frequency into a separate file.

The file names have the format:

```
<noise-plot>_<device-name>_<ob-node-1>_<ob-node-2>_<number>_acgf_des.tdr
```

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Output Data

Table 142 Device noise data

Keyword	Description
eeDiffusionLNS	Electron diffusion LNS
hhDiffusionLNS	Hole diffusion LNS
eeMonopolarGRLNS	Trace of electron monopolar GR LNS
hhMonopolarGRLNS	Trace of hole monopolar GR LNS
eeFlickerGRLNS	Trace of electron flicker GR LNS
hhFlickerGRLNS	Trace of hole flicker GR LNS
ReLNVXVSD	Real/imaginary parts of LNVSD
ImLNVXVSD	
ReLNISD	Real/imaginary parts of local noise current spectral density
ImLNISD	
ReeeLNVXVSD	Real/imaginary parts of LNVSD for electrons
ImeeLNVXVSD	
RehhLNVXVSD	Real/imaginary parts of LNVSD for holes
ImhhLNVXVSD	
ReeeDiffusionLNVXVSD	Real/imaginary parts of diffusion LNVSD for electrons
ImeeDiffusionLNVXVSD	
RehhDiffusionLNVXVSD	Real/imaginary parts of diffusion LNVSD for holes
ImhhDiffusionLNVXVSD	
ReeeMonopolarGRLNVXVSD	Real/imaginary parts of electron monopolar LNVSD
ImeeMonopolarGRLNVXVSD	
RehhMonopolarGRLNVXVSD	Real/imaginary parts of hole monopolar LNVSD
ImhhMonopolarGRLNVXVSD	
ReeeFlickerGRLNVXVSD	Real/imaginary parts of electron flicker GR LNVSD
ImeeFlickerGRLNVXVSD	
RehhFlickerGRLNVXVSD	Real/imaginary parts of hole flicker GR LNVSD
ImhhFlickerGRLNVXVSD	

## Chapter 23: Noise, Fluctuations, and Sensitivity

### Noise Output Data

*Table 142 Device noise data (Continued)*

Keyword	Description
ReTrapLNISD	Real/imaginary parts of local trapping noise current spectral density
ImTrapLNISD	
ReTrapLNVSD	Real/imaginary parts of local trapping noise voltage spectral density
ImTrapLNVSD	
PoECReACGreenFunction	Real/imaginary parts of $G_n$
PoECImACGreenFunction	
CurECReACGreenFunction	
CurECImACGreenFunction	
PoHCReACGreenFunction	Real/imaginary parts of $G_p$
PoHClmACGreenFunction	
CurHCReACGreenFunction	
CurHClmACGreenFunction	
PoETReACGreenFunction	Real/imaginary parts of $G_{T_n}$
PoETImACGreenFunction	
CurETReACGreenFunction	
CurETImACGreenFunction	
PoHTReACGreenFunction	Real/imaginary parts of $G_{T_p}$
PoHTImACGreenFunction	
CurHTReACGreenFunction	
CurHTImACGreenFunction	
PoLTReACGreenFunction	Real/imaginary parts of $G_T$
PoLTImACGreenFunction	
CurLTReACGreenFunction	
CurLTImACGreenFunction	
PoPotReACGreenFunction	Real/imaginary parts of $G_\phi$
PoPotImACGreenFunction	
CurPotReACGreenFunction	
CurPotImACGreenFunction	
PoGeoGreenFunction	Real part of the Green's functions for geometric variations
CurGeoGreenFunction	

## Chapter 23: Noise, Fluctuations, and Sensitivity

### References

Table 142 Device noise data (Continued)

Keyword	Description
GradPoECReACGreenFunction	Real/imaginary parts of $\nabla \cdot G_n$ , $\nabla \cdot G_p$ , $\nabla \cdot G_{T_n}$ , $\nabla \cdot G_{T_p}$ for voltage noise
GradPoECImACGreenFunction	
GradPoHCReACGreenFunction	
GradPoHCImACGreenFunction	
GradPoETReACGreenFunction	
GradPoETImACGreenFunction	
GradPoHTReACGreenFunction	
GradPoHTImACGreenFunction	
Grad2PoECACGreenFunction	$ G_n ^2$ and $ G_p ^2$ for voltage noise
Grad2PoHCACGreenFunction	
AllLNS	All used LNS
AllLNVXVSD	All used LNVSD
GreenFunctions	Green's functions and their gradients

## References

- [1] F. Bonani *et al.*, “An Efficient Approach to Noise Analysis Through Multidimensional Physics-Based Models,” *IEEE Transactions on Electron Devices*, vol. 45, no. 1, pp. 261–269, 1998.
- [2] J.-P. Nougier, “Fluctuations and Noise of Hot Carriers in Semiconductor Materials and Devices,” *IEEE Transactions on Electron Devices*, vol. 41, no. 11, pp. 2034–2049, 1994.
- [3] F. Bonani and G. Ghione, “Generation–recombination noise modelling in semiconductor devices through population or approximate equivalent current density fluctuations,” *Solid-State Electronics*, vol. 43, no 2, pp. 285–295, 1999.

# 24

## Tunneling

---

*This chapter presents the tunneling models available in Sentaurus Device.*

In current microelectronic devices, tunneling has become a very important physical effect. In some devices, tunneling leads to undesired leakage currents (for gates in small MOSFETs). For other devices such as EEPROMs, tunneling is essential for the operation of the device.

The tunneling models discussed in this chapter refer to elastic charge-transport processes at interfaces or contacts. Tunneling also plays a role in some generation–recombination models (see [Chapter 16 on page 473](#)). These models do not deal with spatial transport of charge and, therefore, are not discussed here. In addition to tunneling, hot-carrier injection can also contribute to carrier transport across barriers. To model hot-carrier injection, see [Chapter 25 on page 844](#). Tunneling to traps is discussed in [Nonlocal Tunneling for Traps on page 576](#).

---

### Overview of Tunneling Models

Sentaurus Device provides different tunneling models. The nonlocal tunneling model is the most versatile (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 826](#)) and includes the following features:

- Handles arbitrary barrier shapes
- Includes carrier heating terms
- Allows you to describe tunneling between the valence band and conduction band
- Offers several different approximations for the tunneling probability

Use this model to describe tunneling at Schottky contacts, tunneling in heterostructures, and gate leakage through thin stacked insulators.

## Chapter 24: Tunneling

### Fowler–Nordheim Tunneling

The next most powerful model is the direct tunneling model (see [Direct Tunneling on page 822](#)), and it includes the following features:

- Assumes a trapezoidal barrier (this restricts the range of application to tunneling through insulators)
- Neglects heating of the tunneling carriers
- Optionally, accounts for image charge effects (at the cost of reduced numeric robustness)

Use this model to describe leakage through thin gate insulators, provided those are of uniform or of uniformly graded composition.

The Fowler–Nordheim model is the simplest (see [Fowler–Nordheim Tunneling](#)).

If the simulated device contains a floating gate, gate currents are used to update the charge on the floating gate after each time step in transient simulations. If EEPROM cells are simulated in two dimensions, it is generally necessary to include an additional coupling capacitance between the control and floating gates to account for the additional influence of the third dimension on the capacitance between these electrodes (see [Floating Metal Contacts on page 280](#)). The additional floating gate capacitance can be specified as FGcap in the Electrode statement (see [Physical Models and the Hierarchy of Their Specification on page 66](#)).

---

## Fowler–Nordheim Tunneling

Fowler–Nordheim tunneling is a special case of tunneling also covered by the nonlocal and direct tunneling models, where tunneling is to the conduction band of the oxide. The Fowler–Nordheim tunneling model is simple and efficient, and has proven useful to describe erase operations in EEPROMs, which is the application for which this model is recommended.

---

## Using Fowler–Nordheim

To switch on the Fowler–Nordheim tunneling model, specify `Fowler` as an option to `GateCurrent` in an appropriate interface `Physics` section, as follows:

```
Physics(MaterialInterface="Silicon/Oxide") { GateCurrent(Fowler) }
```

or:

```
Physics(MaterialInterface="Silicon/Oxide") { GateCurrent(Fowler(EVB)) }
```

The second specification activates the tunneling of electrons from and to the valence band as discussed in [Fowler–Nordheim Model on page 821](#).

## Chapter 24: Tunneling

### Fowler–Nordheim Tunneling

If `GateCurrent` is specified as above, Sentaurus Device computes gate currents between all silicon–oxide interfaces and the electrodes. The computation can be restricted to selected interfaces using `region-interface Physics` sections.

For simple one-gate devices, the tunneling current can be monitored on a contact specified by the keyword `GateName` in the `GateCurrent` statement.

You can use the Fowler–Nordheim tunneling model with any of the hot-carrier injection models (see [Chapter 25 on page 844](#)). The following example switches on the Fowler–Nordheim tunneling model and the classical lucky electron injection model simultaneously:

```
GateCurrent( Fowler Lucky )
```

---

## Fowler–Nordheim Model

The Fowler–Nordheim model reads:

$$j_{FN} = AF_{ins}^2 \exp -\frac{B}{F_{ins}} \quad (806)$$

where  $j_{FN}$  is the tunnel current density,  $F_{ins}$  is the insulator electric field at the interface, and  $A$  and  $B$  are physical constants. The electric field at the interface shown by Sentaurus Visual is an interpolation of the fields in both regions, but Sentaurus Device uses the insulator field internally.

Due to the large energy difference between oxide and silicon conduction bands, tunneling electrons create electron–hole pairs in the erase operation when they enter the semiconductor. This additional carrier generation is included by an energy-independent multiplication factor  $\gamma > 1$  :

$$j_n = \gamma \cdot j_{FN} \quad (807)$$

$$j_p = (\gamma - 1) \cdot j_{FN} \quad (808)$$

If the electrons flow in an opposite direction, by default,  $\gamma = 1$ . The formulas above reflect the default behavior of Sentaurus Device, but sometimes the Fowler–Nordheim equation is used to emulate other tunneling effects, for example, the tunneling of electrons from the valence band into the gate. Such capability is activated by the additional keyword `EVB` in the command file. Sentaurus Device will continue to function even if  $\gamma < 1$ . For any electron tunneling direction, particularly a floating body SOI, this tunneling current is important because it strongly defines floating body potential. Different coefficients are needed for the write and erase operations because, in the first case, the electrons are emitted from monocrystalline silicon and, in the latter case, they are emitted from the polysilicon contact into the oxide.

The tunneling current is implemented as a current boundary condition at the interface where the current is produced.

## Chapter 24: Tunneling

### Direct Tunneling

For transient simulations, if the `Interface` keyword is also present in the `GateCurrent` section, carrier tunneling with explicitly evaluated boundary conditions for continuity equations is activated (similar to carrier injection with explicitly evaluated boundary conditions for continuity equations in [Carrier Injection With Explicitly Evaluated Boundary Conditions for Continuity Equations on page 867](#)).

---

## Fowler–Nordheim Parameters

The parameters of the Fowler–Nordheim model can be modified in the `FowlerModel` parameter set. Sentaurus Device uses different parameters (denoted as `erase` and `write`) depending on the direction of the electric field between the contact and semiconductor in the oxide layer. For example, if the field points from the contact to the semiconductor (that is, electrons flow into the contact), the ‘`write`’ parameter set is used. [Table 143](#) lists the parameters and their default values.

*Table 143 Coefficients for Fowler–Nordheim tunneling (defaults for silicon–oxide interface)*

Symbol	Parameter name	Default value	Unit	Remarks
A (erase)	<code>Ae</code>	$1.87 \times 10^{-7}$	A/V <sup>2</sup>	A for the erase cycle
B (erase)	<code>Be</code>	$1.88 \times 10^8$	V/cm	B for the erase cycle
A (write)	<code>Aw</code>	$1.23 \times 10^{-6}$	A/V <sup>2</sup>	A for the write cycle
B (write)	<code>Bw</code>	$2.37 \times 10^8$	V/cm	B for the write cycle
$\gamma$	<code>Gm</code>	1	1	

---

## Direct Tunneling

Direct tunneling is the main gate leakage mechanism for oxides thinner than 3 nm. It turns into Fowler–Nordheim tunneling at oxide fields higher than approximately 6 MV/cm, independent of the oxide thickness. This section describes a fully quantum-mechanical tunneling model that is restricted to trapezoidal tunneling barriers and covers both direct tunneling and the Fowler–Nordheim regime. Optionally, the model considers the reduction of the tunneling barrier due to image forces.

A variant of this direct tunneling model is used to describe spin-selective tunneling through magnetic tunnel junctions (see [Transport Through Magnetic Tunnel Junctions on page 922](#)).

## Chapter 24: Tunneling

### Direct Tunneling

---

## Using Direct Tunneling

The direct tunneling model is specified as an option of the `GateCurrent` statement (see [Using Fowler–Nordheim on page 820](#)) in an appropriate interface `Physics` section:

```
Physics(MaterialInterface="Silicon/Oxide") {  
    GateCurrent( DirectTunneling )  
}
```

The keyword `GateName` and the compatibility with the hot-carrier injection models (see [Chapter 25 on page 844](#)) are as for the Fowler–Nordheim model, see [Fowler–Nordheim Tunneling on page 820](#).

To switch on the image force effect, the parameters `E0`, `E1`, and `E2` must be specified (in eV) in the parameter file. If these values are all equal (the default), the image force is neglected. Recommended values for both electrons and holes are `E0=0`, `E1=0.3`, and `E2=0.7`. Including the image force effect degrades convergence. For most purposes, the effect can be approximated by reducing the overall barrier height. For more information on model parameters, see [Direct Tunneling Parameters on page 825](#).

To plot the direct tunneling current, the keywords `eSchenkTunnel` or `hSchenkTunnel` must be included in the `Plot` section:

```
Plot { eSchenkTunnel hSchenkTunnel }
```

---

## Direct Tunneling Model

This section summarizes the model described in the literature [1]. It presents the formulas for electrons only; the hole expressions are analogous. The electron tunneling current density is:

$$j_n = \frac{qm_C k}{2\pi^2 \hbar^3} \int_0^\infty dE Y(E)^- T(0) \ln \exp \left[ \frac{E_{F,n}(0) - E_C(0) - E}{kT(0)} \right] + 1 \\ - T(d) \ln \exp \left[ \frac{E_{F,n}(d) - E_C(0) - E}{kT(d)} \right] + 1 \quad (809)$$

where  $d$  is the effective thickness of the barrier,  $m_C$  is a mass prefactor, the argument 0 denotes one (the ‘substrate’) side of the barrier and  $d$  denotes the other (‘gate’) side, and  $E$  is the energy of the elastic tunnel process (relative to  $E_C(0)$ ).

## Chapter 24: Tunneling

### Direct Tunneling

$\Upsilon(E) = 2/(1 + g(E))$  is the transmission coefficient for a trapezoidal potential barrier, with:

$$g(E) = \frac{\pi^2}{2} \sqrt{\frac{E_T}{E}} \sqrt{\frac{m_{Si}}{m_G}} (\text{Bi}'_d \text{Ai}_0 - \text{Ai}'_d \text{Bi}_0)^2 + \sqrt{\frac{E}{E_T}} \sqrt{\frac{m_G}{m_{Si}}} (\text{Bi}_d \text{Ai}'_0 - \text{Ai}_d \text{Bi}'_0)^2 + \frac{\sqrt{m_G m_{Si}} \hbar \Theta_{ox}}{m_{ox} \sqrt{E E_T}} (\text{Bi}'_d \text{Ai}'_0 - \text{Ai}'_d \text{Bi}'_0)^2 + \frac{m_{ox}}{\sqrt{m_G m_{Si}}} \frac{\sqrt{E E_T}}{\hbar \Theta_{ox}} (\text{Bi}_d \text{Ai}_0 - \text{Ai}_d \text{Bi}_0) \quad (810)$$

and:

$$\text{Ai}_0 = \text{Ai} \left[ \frac{E_B(E) - E}{\hbar \Theta_{ox}} \right], \quad \text{Ai}_d = \text{Ai} \left[ \frac{E_B(E) - qF_{ox}d - E}{\hbar \Theta_{ox}} \right] \quad (811)$$

and so on, where  $\hbar \Theta_{ox} = (q^2 \hbar^2 F_{ox}^2 / 2m_{ox})^{1/3}$  and  $E_B(E)$  denotes the (substrate-side) barrier height for electrons of energy  $E$ .  $E_T$  is the tunneling energy with respect to the conduction band edge on the gate side,  $E_T = E - E_C(d) - E_C(0)$ .

For compatibility with an earlier implementation of the model,  $E_T$  is truncated to the value specified by the parameter `E_F_M` if it exceeds this value.  $F_{ox} = V_{ox}/d$  is the electric field in the oxide (assumed to be uniform within the oxide, and including a band edge-related term when different barrier heights are specified at the two insulator interfaces). The quantities  $m_G$ ,  $m_{ox}$ , and  $m_{Si}$  represent the electron masses in the three materials, respectively.  $\text{Ai}$  and  $\text{Bi}$  are Airy functions, and  $\text{Ai}'$  and  $\text{Bi}'$  are their derivatives.

## Image Force Effect

Ultrathin oxide barriers are affected by the image force effect. If the latter is neglected,  $E_B(E)$  is the bare barrier height  $E_B$ , which is an input parameter. The image force effect is included in the model by taking  $E_B(E)$  as an energy-dependent pseudobarrier:

$$E_B(E) = E_B(E_0) + \frac{E_B(E_2) - E_B(E_0)}{(E_2 - E_0)(E_1 - E_2)} (E - E_0)(E_1 - E) - \frac{E_B(E_1) - E_B(E_0)}{(E_1 - E_0)(E_1 - E_2)} (E - E_0)(E_2 - E) \quad (812)$$

where  $E_0$ ,  $E_1$ , and  $E_2$  are chosen in the lower energy range of the barrier potential (between 0 eV and 1.5 eV in practical cases). If these values are chosen to be equal, the image force effect is automatically switched off. Otherwise, for each bias point:

$$S_{tra}(E) = S_{im}(E) \quad (813)$$

is solved for  $E_j$  ( $j = 0, 1, 2$ ), which results in three pseudobarrier heights  $E_B(E_j)$  used in Equation 812. In Equation 813,  $S_{tra}(E)$  is the action of the trapezoidal pseudobarrier:

$$S_{tra}(E) = \frac{2}{3} \left| \left[ \frac{E_B(E) - qF_{ox}d - E}{\hbar \Theta_{ox}} \right]^{\frac{3}{2}} \Theta [E_B(E) - qF_{ox}d - E] - \left[ \frac{E_B(E) - E}{\hbar \Theta_{ox}} \right]^{\frac{3}{2}} \right| \quad (814)$$

## Chapter 24: Tunneling

### Direct Tunneling

and  $S_{\text{im}}(E)$  is the action of the respective image potential barrier:

$$S_{\text{im}}(E) = \sqrt{\frac{2m_{\text{ox}}}{\hbar^2}} \int_{x_l(E)}^{x_r(E)} d\xi \sqrt{E_B - qF_{\text{ox}}\xi + E_{\text{im}}(\xi) - E} \quad (815)$$

$x_{l,r}(E)$  denotes the classical turning points for a given carrier energy that follow from:

$$E_T - qF_{\text{ox}}x_{l,r} + E_{\text{im}}(x_{l,r}) = E \quad (816)$$

For the thickness of the pseudobarrier,  $d = x_r(0) - x_l(0)$  is used, that is, the distance between the two turning points at the energy of the semiconductor conduction band edge of the interface. Therefore,  $d$  is smaller than the user-given oxide thickness, when the image force effect is considered. The image potential itself is given by:

$$E_{\text{im}}(x) = \frac{q^2}{16\pi\epsilon_{\text{ox}}} \sum_{n=0}^{10} (k_1 k_2)^n \left[ \frac{k_1}{nd+x} + \frac{k_2}{d(n+1)-x} + \frac{2k_1 k_2}{d(n+1)} \right] \quad (817)$$

with:

$$k_1 = \frac{\epsilon_{\text{ox}} - \epsilon_{\text{Si}}}{\epsilon_{\text{ox}} + \epsilon_{\text{Si}}}, \quad k_2 = \frac{\epsilon_{\text{ox}} - \epsilon_{\text{G}}}{\epsilon_{\text{ox}} + \epsilon_{\text{G}}} = -1 \quad (818)$$

In Equation 818,  $\epsilon_{\text{ox}}$ ,  $\epsilon_{\text{Si}}$ , and  $\epsilon_{\text{G}}$  denote the dielectric constants for the oxide, substrate, and gate material, respectively.

## Direct Tunneling Parameters

The parameters of the direct tunneling model are modified in the `DirectTunneling` parameter set. The appropriate default parameters for an oxide barrier on silicon are in [Table 144](#). The parameters are specified according to the interface.

*Table 144 Coefficients for direct tunneling (defaults for oxide barrier on silicon)*

Symbol	Parameter name	Electrons	Holes	Unit	Description
$\epsilon_{\text{ox}}$	<code>eps_ins</code>	2.13	2.13	1	Optical dielectric constant
$E_F(d)$	<code>E_F_M</code>	11.7	11.7	eV	Fermi energy for gate contact
$m_G$	<code>m_M</code>	1	1	$m_0$	Effective mass in gate contact
$m_{\text{ox}}$	<code>m_ins</code>	0.50	0.77	$m_0$	Effective mass in insulator
$m_{\text{Si}}$	<code>m_s</code>	0.19	0.16	$m_0$	Effective mass in substrate
$m_C$	<code>m_dos</code>	0.32	0	$m_0$	Semiconductor DOS effective mass

## Chapter 24: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

Table 144 Coefficients for direct tunneling (defaults for oxide barrier on silicon) (Continued)

Symbol	Parameter name	Electrons	Holes	Unit	Description
$E_B$	<code>E_barrier</code>	3.15	4.73	eV	Semiconductor/insulator barrier (no image force)
$E_0, E_1, E_2$	<code>E0, E1, E2</code>	0	0	eV	Energy nodes 0, 1, and 2 for pseudobarrier calculation

If tunneling occurs between two semiconductors, the coefficients for metal (`m_M` and `E_F_M`) are not used. The semiconductor parameters for the respective interface are used. It is not valid to have contradicting parameter sets for two interfaces of an insulator between which tunneling occurs. In particular, `eps_ins`, `m_ins`, `E0`, `E1`, and `E2` must agree in the parameter specifications for both interfaces.

---

## Nonlocal Tunneling at Interfaces, Contacts, and Junctions

The tunneling current depends on the band edge profile along the entire path between the points connected by tunneling. This makes tunneling a nonlocal process. In general, the band edge profile has a complicated shape, and Sentaurus Device must compute it by solving the transport equations and the Poisson equation. The model described here takes this dependence fully into account.

To use the nonlocal tunneling model:

1. Construct a special-purpose nonlocal mesh (see [Defining Nonlocal Meshes](#)).
2. Specify the physical details of the tunneling model (see [Specifying Nonlocal Tunneling Model on page 828](#)).
3. Adjust the physical and numeric parameters (see [Nonlocal Tunneling Parameters on page 830](#)).

---

## Defining Nonlocal Meshes

It is necessary to specify a special-purpose nonlocal mesh for each part of the device for which you want to use the nonlocal tunneling model. Nonlocal meshes consist of ‘nonlocal’ lines that represent the tunneling paths for the carriers.

To control the construction of the nonlocal mesh, use the options of the keyword `NonLocal` in the global `Math` section.

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

Nonlocal meshes can be constructed using different forms:

- In the simpler form, `NonLocal` specifies barrier regions.

The following example constructs a nonlocal mesh for a tunneling barrier that consists of the regions `oxtop` and `oxbottom`:

```
Math { Nonlocal "NLM" ( Barrier(Region="oxtop" Region="oxbottom") ) }
```

See [Specification Using Barrier on page 205](#).

- In the more general form, `NonLocal` specifies an interface or a contact where the mesh is constructed, and a distance `Length` (given in centimeters). Sentaurus Device then connects semiconductor vertices with a distance from the interface or contact no larger than `Length` to the interface or contact. These connections form the centerpiece of the nonlocal lines.

In the following example, Sentaurus Device constructs a nonlocal mesh called `"NLM"` that connects vertices up to a distance of 5 nm to the `Gate` electrode:

```
Math { Nonlocal "NLM" (Electrode="Gate" Length=5e-7) }
```

See [Specification Using a Reference Surface on page 205](#).

Sentaurus Device introduces a coordinate along each nonlocal line. The interface or contact is at coordinate zero; the vertex for which the nonlocal line is constructed is at a positive coordinate. Below, the terms *upper* and *lower* refer to the orientation according to these nonlocal line-specific coordinates. This orientation is not related to the orientation of the mesh axes.

To form the nonlocal lines, Sentaurus Device extends the connection at the upper end, to fully cover the box for the vertex that is connected. Optionally, for nonlocal meshes at interfaces, the connection can be extended at the lower end (beyond the interface) by an amount specified by `Permeation` (in centimeters).

Sentaurus Device handles each nonlocal line separately. Therefore, two nonlocal lines connecting two vertices to the same interface do not allow the computing of tunneling between these vertices. To compute tunneling between vertices on the two sides of an interface, they must be covered by a single nonlocal line. This is where `Permeation` is needed.

For nonlocal meshes not used for trap tunneling, `Permeation` also affects the integration range for tunneling. With zero `Permeation`, the lower endpoint for tunneling is always at the interface; for positive `Permeation`, it can be anywhere on the nonlocal line. Therefore, for tunneling at smooth barriers (for example, band-to-band tunneling at a steep p-n junction), the nonlocal mesh used should not be reused for trap tunneling (see [Nonlocal Tunneling for Traps on page 576](#)), and `Permeation` should be positive.

When using the more general form of nonlocal mesh construction, specify a nonlocal mesh for only one side of a tunneling barrier. If on one side of the tunneling barrier there is a

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

contact or a metal–nonmetal interface, specify the mesh there. In other cases, specify the nonlocal mesh for the interface from which the carriers will tunnel away. If this side can change during the simulation, specify a nonzero `Permeation` (approximately as much as `Length` exceeds the barrier thickness). For tunneling barriers with multiple layers, do not specify a nonlocal mesh for the interfaces internal to the barrier.

For more options to the keyword `NonLocal`, see [Table 226 on page 1612](#). For details about the nonlocal mesh and its construction, see [Constructing Nonlocal Meshes on page 204](#).

#### Note:

The nonlocality of tunneling can increase dramatically the time that is needed to solve the linear systems in the Newton iteration. To limit the speed degradation, keep `Length` and `Permeation` as small as possible. Consider optimizing the nonlocal mesh using the advanced options of `NonLocal` (see [Constructing Nonlocal Meshes on page 204](#)).

---

## Specifying Nonlocal Tunneling Model

The nonlocal tunneling model is activated and controlled in the global `Physics` section. Each tunneling event connects two points on a nonlocal line. Sentaurus Device distinguishes between tunneling to the conduction band and to the valence band at the lower of the two points.

To switch on these terms, use the options `eBarrierTunneling` and `hBarrierTunneling`. For example:

```
Physics {
    eBarrierTunneling "NLM" hBarrierTunneling "NLM"
}
```

switches on electron and hole tunneling for the nonlocal mesh called "`NLM`", and:

```
Physics {
    eBarrierTunneling "NLM" (PeltierHeat)
}
```

switches on tunneling to the conduction band only and activates the Peltier heating of the tunneling particles.

By default, all tunneling to the conduction band at the lower point on the line originates from the conduction band at the upper point,  $j_C = j_{CC}$  ('electron tunneling'), see [Equation 829](#). Likewise, by default, the tunneling to the valence band at the lower point originates from the valence band at the upper point,  $j_V = j_{VV}$  (that is, 'hole tunneling').

In addition, Sentaurus Device supports nonlocal band-to-band tunneling. To include the contributions by tunneling to and from the valence band at the upper point in  $j_C$ , specify `eBarrierTunneling` with the `Band2Band=Simple` option. Then,  $j_C = j_{CC} + j_{CV}$ , see [Equation 829](#) and [Equation 832](#). To include contributions by tunneling to and from the

## Chapter 24: Tunneling

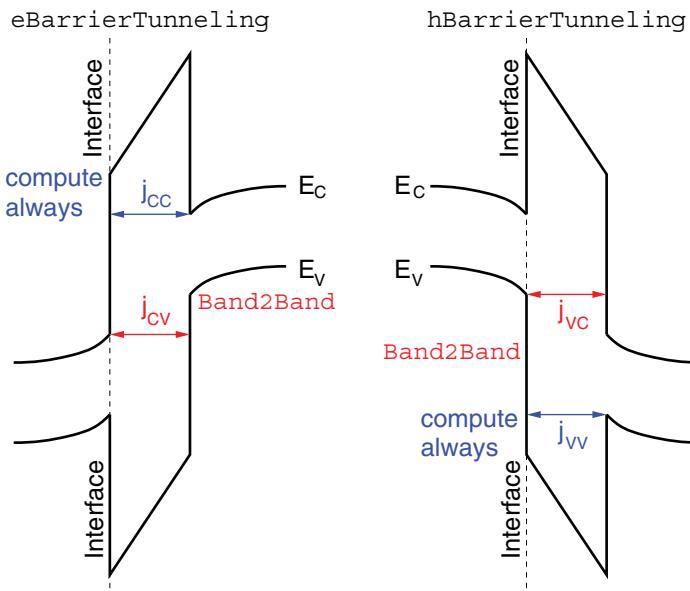
### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

conduction band at the upper point to  $j_V$ , specify `hBarrierTunneling` with the `Band2Band=Simple` option. Then,  $j_V = j_{VC} + j_{VV}$ . With `Band2Band=Full`, the nonlocal band-to-band tunneling model uses [Equation 499](#) instead of [Equation 831](#) to calculate the band-to-band tunneling contribution, which makes the model consistent with the [Dynamic Nonlocal Path Band-to-Band Tunneling Model on page 529](#) provided that the dynamic nonlocal path band-to-band model uses the direct tunneling model with the Franz dispersion relation. With `Band2Band=UpsideDown` (or only `Band2Band`), an obsolete, physically flawed, band-to-band tunneling model is activated.

For backward compatibility, you can activate the nonlocal tunneling model in an interface-specific or contact-specific `Physics` section; in that case, specify `eBarrierTunneling` and `hBarrierTunneling` as options to `Recombination`, and omit the nonlocal mesh name. The specification applies to an unnamed nonlocal mesh that has been constructed for the same location (see [Unnamed Meshes on page 208](#)).

[Figure 50](#) illustrates the four contributions to the total tunneling current.

*Figure 50 Various nonlocal tunneling current contributions*



By default, Sentaurus Device assumes a single-band parabolic band structure for the tunneling particles, uses a WKB-based model for the tunneling probability, and ignores band-to-band tunneling and Peltier heating. Options to `eBarrierTunneling` and `hBarrierTunneling` override the default behavior. For available options, see [Table 242 on page 1634](#). For a detailed discussion of the physics of the nonlocal tunneling model, see [Physics of Nonlocal Tunneling Model on page 832](#).

When specifying the `Multivalley` option, the tunneling model uses the multivalley band structure (see [Multivalley Band Structure on page 327](#)). This allows you to account for

## Chapter 24: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

conduction and valence bands with multiple valleys, anisotropic masses, and possibly geometric confinement. See [Multivalley Band Structure and Geometric Quantization on page 837](#).

When specifying the `BarrierLowering` option, the tunneling model accounts for position-dependent barrier lowering corrections in the conduction-band and valence-band edges, shifting them inwards, toward the midgap. See [WKB Tunneling Probability on page 833](#) and [Equation 825 on page 835](#).

---

## Nonlocal Tunneling Parameters

The nonlocal tunneling model has several fit parameters. They are specified in the `BarrierTunneling` parameter set.

The parameter pair `g` determines the dimensionless prefactors  $g_C$  and  $g_V$  (see [Equation 828](#) and [Equation 831](#)). For unnamed meshes, specify them in the parameter set that is specific to the contact or interface for which the nonlocal tunneling model is activated. For named meshes, specify them in the global parameter set.

The parameter pair `mt` determines the tunneling masses  $m_C$  and  $m_V$  (see [Equation 819](#) and [Equation 820](#)). They are properties of the materials that form the tunneling barrier. Therefore, specify them (in units of  $m_0$ ) in region-specific or material-specific parameter sets. For tunneling at contacts, also specify the masses for the contact parameter set when using `Transmission` or `Schroedinger` (see [Equation 824 on page 834](#) and [Schrödinger Equation-Based Tunneling Probability on page 836](#)).

Sentaurus Device treats effective tunneling masses of value zero as undefined. If an effective tunneling mass for a region is undefined, Sentaurus Device uses the effective tunneling mass for the interface or contact for which tunneling is activated (for unnamed meshes) or the tunneling mass from the global parameter set (for named meshes). By default, all effective tunneling masses are undefined. Finally, if the model is used with the `Multivalley` option (see [Multivalley Band Structure and Geometric Quantization on page 837](#)), masses extracted from the multivalley model override the tunneling masses wherever they are available.

The following example changes the prefactors  $g_C$  and  $g_V$  to 1 and 2, respectively:

```
BarrierTunneling {  
    mt = 0.5, 0.5  
    g = 1 , 2  
}  
Material = "Oxide" {  
    BarrierTunneling {  
        mt = 0.42 , 1.0  
    }  
}
```

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

The tunneling masses are set to  $0.5m_0$ . In oxide, it sets the tunneling masses  $m_C$  and  $m_V$  to 0.42 and 1, respectively; these values take precedence over the masses specified globally.

The parameters `eoffset` and `hoffset` are lists of nonnegative energy shifts (in eV) that are added to the conduction-band and valence-band edges, shifting them outwards, away from the midgap. Specify them in a region-specific or material-specific parameter set. By default, a single shift value of zero is assumed. Sentaurus Device computes the tunneling current as the sum of the currents for each shift. Shifts in different regions are combined according to the position where they appear in the list; if the lists in different regions have different lengths, the last value in the shorter lists are repeated to extend them to the length of the longest list.

The parameter pair `alpha` determines the fit factors  $\alpha_n$  and  $\alpha_p$  for the quantization corrections for tunneling from inversion layers through insulator barriers (see [Density Gradient Quantization Correction on page 837](#)). The default values are zero, disabling the corrections. To enable them, set the parameters to one (or another positive value) in the interface- or contact-specific parameter set (for unnamed meshes) or in the global parameter set (for named meshes).

The parameter pairs `QuantumPotentialFactor` and `QuantumPotentialPosFac` specify the prefactors for the electron and hole quantum potentials obtained from the density gradient model (see [Density Gradient Model on page 362](#)) that can be added to the band edges used for evaluating the tunneling rate. By default, the parameters are zero, such that the quantum potentials are neglected in the computation of tunneling. When they are nonzero, the quantum potentials multiplied by the corresponding prefactors are added to the conduction and valence band edges for the computation of tunneling current.

For `QuantumPotentialFactor`, an addition is performed irrespective of the sign of the quantum potential. For `QuantumPotentialPosFac`, an addition is performed only where the quantum potential is positive, that is, only where quantization causes an effective widening of the band gap.

The quantum corrections activated by giving `QuantumPotentialFactor` or `QuantumPotentialPosFac` a nonzero value are intended for situations where quantization is caused by confinement perpendicular to the tunneling direction. In contrast, `alpha` is intended for tunneling from an inversion layer, where quantization is due to confinement in the tunneling direction.

`BarrierTunneling` can also be applied specifically to named nonlocal meshes. The following example changes the oxide tunneling mass for the nonlocal mesh `NLM` only:

```
Material = "Oxide" {
    BarrierTunneling "NLM" {
        mt = 0.42 , 1.0
    }
}
```

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

Specifications for named nonlocal meshes take precedence over the general specifications.

Specify numeric parameters for the model in the `Math` section, as an option of `NonLocal` for a given nonlocal mesh. The parameter `Digits` determines the relative accuracy (the number of valid decimal digits) to which Sentaurus Device computes the integrals in [Equation 829](#) and [Equation 832](#). The default value is `2`. The parameter `EnergyResolution` (given in eV) is a lower limit for the energy step that Sentaurus Device uses to perform the integrations and it defaults to `0.005`. The purpose of `EnergyResolution` is to limit the runtime for computing the tunneling currents in case the value of `Digits` is too large.

The following example increases the energy resolution for tunneling on the nonlocal mesh "NLM" to 1 meV and the relative accuracy of the tunneling current computation to three digits:

```
Math {  
    NonLocal "NLM" (  
        ...  
        Digits=3  
        EnergyResolution=0.001  
    )  
}
```

---

## Visualizing Nonlocal Tunneling

To visualize nonlocal tunneling, specify the keyword `eBarrierTunneling` or `hBarrierTunneling` in the `Plot` section (see [Device Plots on page 177](#)). The quantities plotted are in units of  $\text{cm}^{-3}\text{s}^{-1}$  and represent the rate at which electrons and holes are generated or disappear due to tunneling. To plot the Peltier heat generated in the conduction band and valence band due to nonlocal tunneling, specify `eNLLTunnelingPeltierHeat` and `hNLLTunnelingPeltierHeat`. The Peltier heat is plotted in units of  $\text{Wcm}^{-3}$  and is available regardless of whether it is accounted for in the temperature equation.

The rates are plotted vertexwise and are averaged over the semiconductor volume controlled by a vertex. Therefore, they depend on the mesh spacing. This dependence can become particularly strong at interfaces where the band edges change abruptly.

---

## Physics of Nonlocal Tunneling Model

The nonlocal tunneling model in Sentaurus Device is based on the approach presented in the literature [\[2\]](#), but provides significant enhancements over the model presented there.

## Chapter 24: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

### WKB Tunneling Probability

The computation of the tunneling probabilities  $\Gamma_{C,v}$  (for carriers tunneling to the  $v$ -th shifted conduction band at the interface or contact) and  $\Gamma_{V,v}$  (for tunneling to the  $v$ -th shifted valence band) is, by default, based on the WKB approximation.

The WKB approximation uses the local (imaginary) wave numbers of particles at position  $r$  and with energy  $\varepsilon$ :

$$\kappa_{C,v}(r, \varepsilon) = \sqrt{2m_C(r)[E_{C,v}(r) - \varepsilon]} \Theta[E_{C,v}(r) - \varepsilon]/\hbar \quad (819)$$

$$\kappa_{V,v}(r, \varepsilon) = \sqrt{2m_V(r)[\varepsilon - E_{V,v}(r)]} \Theta[\varepsilon - E_{V,v}(r)]/\hbar \quad (820)$$

Here,  $m_C$  is the conduction-band tunneling mass and  $m_V$  is the valence-band tunneling mass.

Both tunneling masses are adjustable parameters (see [Nonlocal Tunneling Parameters on page 830](#)).  $E_{C,v}$  and  $E_{V,v}$  are the conduction and valence bands energies shifted by the  $v$ -th value in `eoffset` and `hoffset` (see [Nonlocal Tunneling Parameters on page 830](#)).

Using the local wave numbers and the interface transmission coefficients  $T_{CC,v}$  and  $T_{VV,v}$ , the tunneling probability between positions  $l$  and  $u > l$  for a particle with energy  $\varepsilon$  can be written as:

$$\Gamma_{CC,v}(u, l, \varepsilon) = T_{CC,v}(l, \varepsilon) \exp \int_l^u -2 \kappa_{C,v}(r, \varepsilon) dr \quad T_{CC,v}(u, \varepsilon) \quad (821)$$

and:

$$\Gamma_{VV,v}(u, l, \varepsilon) = T_{VV,v}(l, \varepsilon) \exp \int_l^u -2 \kappa_{V,v}(r, \varepsilon) dr \quad T_{VV,v}(u, \varepsilon) \quad (822)$$

If the option `TwoBand` is specified to `eBarrierTunneling` (see [Table 242 on page 1634](#)), Sentaurus Device replaces  $\kappa_{C,v}$  in [Equation 821](#) with the two-band dispersion relation:

$$\kappa_v = \frac{\kappa_{C,v}\kappa_{V,v}}{\sqrt{\kappa_{C,v}^2 + \kappa_{V,v}^2}} \quad (823)$$

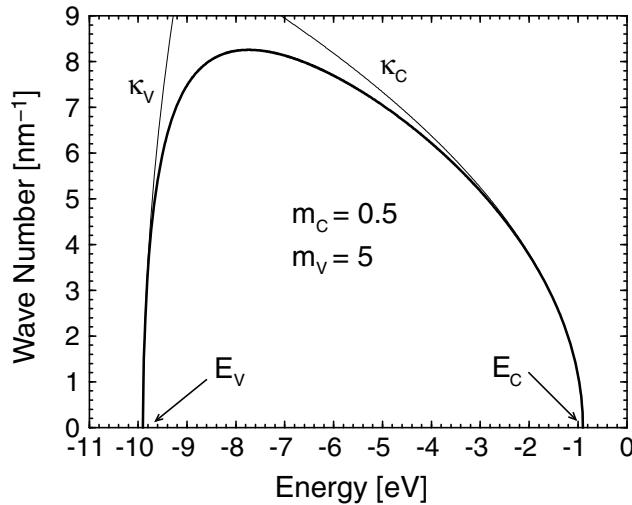
If the option `TwoBand` is specified to `hBarrierTunneling`, Sentaurus Device replaces  $\kappa_{V,v}$  in [Equation 822](#) with the two-band relation [Equation 823](#). Near the conduction and the valence band edge, the two-band dispersion relation approaches the single band dispersion relations [Equation 819](#) and [Equation 820](#), and provides a smooth interpolation in between. [Figure 51](#) illustrates this.

The two-band dispersion relation does not distinguish between electrons and holes. In particular, for the two-band dispersion relation,  $\Gamma_{CC,v} = \Gamma_{VV,v}$ .

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

Figure 51 Comparison of two-band and single-band dispersion relations



The two-band dispersion relation is most useful when band-to-band tunneling is active (keyword `Band2Band`, see [Table 242 on page 1634](#)). However, the two-band dispersion relation can be used independently from band-to-band tunneling.

By default, the interface transmission coefficients  $T_{CC,v}$  and  $T_{VV,v}$  in [Equation 821](#) and [Equation 822](#) equal one. If the `Transmission` option is specified to `eBarrierTunneling` [3]:

$$T_{CC,v}(x, \varepsilon) = \frac{v_{-,v}(x, \varepsilon) \sqrt{v_{-,v}(x, \varepsilon)^2 + 16v_{+,v}(x, \varepsilon)^2}}{v_{+,v}(x, \varepsilon)^2 + v_{-,v}(x, \varepsilon)^2} \quad (824)$$

Here,  $v_{-,v}(x, \varepsilon)$  denotes the velocity of a particle with energy  $\varepsilon$  on the side of the interface or contact at position  $x$  where the particle moves freely in the conduction band, and  $v_{+,v}(x, \varepsilon)$  denotes the imaginary velocity on the side of the tunneling barrier (where the particle is in the gap). If the particle is free or in the barrier on both sides,  $T_{CC,v}(x, \varepsilon) = 1$ . Velocities are the derivatives of the particle energy with respect to the wave number,  $v_v = |\partial\varepsilon/\partial\hbar\kappa_{C,v}|$ . With the `TwoBand` option,  $v_{+,v} = |\partial\varepsilon/\partial\hbar\kappa_v|$  [4] (see [Equation 823](#)). If the `Transmission` option is specified to `hBarrierTunneling`, analogous expressions hold for  $T_{VV,v}$ .

By default, the band-to-band tunneling probabilities  $\Gamma_{CV,v}$  and  $\Gamma_{VC,v}$  are also given by the expressions [Equation 821](#) and [Equation 822](#), respectively. When the `TwoBand` and `Transmission` options are specified for `eBarrierTunneling` to compute  $\Gamma_{CV,v}$ ,  $v_{-,v}$  in the expression for  $T_{CC,v}(r, \varepsilon)$  (see [Equation 824](#)) is the velocity of the particle in the valence band and is computed from valence-band parameters. The converse holds for  $\Gamma_{VC,v}$  when those options are specified for `hBarrierTunneling`.

For metals and contacts, the band-edge energy to compute the interface transmission coefficients is obtained from the `FermiEnergy` parameter of the `BandGap` section for the

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

metal or contact. The masses used to compute the velocities are the tunneling effective masses for the metal or contact.

#### Note:

The default contact material is polysilicon, that is, a semiconductor with a band gap. The conduction band edge of the contact might be energetically close to the valence band of the device semiconductor. If you use `eBarrierTunneling` with the option `Band2Band`, then a negative differential tunneling current might occur. See [Chapter 10 on page 258](#) to change the contact properties, for example, to use metal contacts.

For metals and contacts when the `BarrierLowering` option is specified for `BarrierTunneling`, the band edges are corrected by a position-dependent lowering potential given by:

$$\Delta\Phi_B(r) = \min \frac{Aq}{16\pi\epsilon r}, BL_{max} \quad (825)$$

where  $\epsilon$  is the material dielectric constant,  $r$  is the distance from the interface or contact, and  $A$  is a dimensionless fitting parameter. Because [Equation 825](#) has a singularity for  $r \rightarrow 0$ , Sentaurus Device limits  $r$  by imposing a minimum value. You can set the minimum value for  $r$  by specifying a maximum value for  $\Delta\Phi_B(r)$  using the parameter `BL_max` (see [Nonlocal Tunneling Parameters on page 830](#)). `BL_max` must be positive.

The regionwise barrier-lowering fitting factor  $A$  can be specified by the `A` parameter (default is 1):

```
Material = "Silicon" {
    BarrierTunneling {
        A = 1.02 , 1.0
    }
}
```

The barrier-lowering parameter `BL_max` allows you to adjust the maximum of the barrier lowering (corresponding to the minimum-allowed distance to the contact or interface). Its default value is 0.3 eV.

When the `BarrierLowering` model is defined, the effective band edge is typically lower than the original band edge. For the energy range between the lowered band edge and the original band edge, a thermionic emission-like current is calculated with transmission equal to 1.

To better resolve the lowering potential, an adaptive nonlocal mesh is generated when the `BarrierLowering` model is activated in tunneling. You can control the adaptive mesh by using the following syntax:

```
Math { NonLocal "NLM" (RefinedIntervals=1e-8) }
```

## Chapter 24: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

A smaller `RefinedIntervals` gives a finer mesh. It corresponds to the position-dependent lowering potential defined in [Equation 825](#). Its default value is 1, which produces the same results as previous releases. To have a better refinement, `RefinedIntervals=1e-8` is suggested, corresponding to `A=1` and `BL_max=0.3eV`. With a smaller `A` or larger `BL_max`, the value of `RefinedIntervals` must be smaller.

### Note:

This feature is available only for the `BarrierLowering` model. If the `BarrierLowering` model is not specified, this feature is not activated.

To visualize the `BarrierLowering` model and the modified band edges, you can specify the following in the `NonLocalPlot` section:

```
NonLocalPlot((0 0)){  
    CBBARRIER LOWERING VBBARRIER LOWERING  
    CBEnergywithLowering VBEnergywithLowering  
}
```

where:

- `CBBARRIER LOWERING` and `VBBARRIER LOWERING` are the `BarrierLowering` values of the conduction band and the valence band, respectively.
- `CBEnergywithLowering` and `VBEnergywithLowering` are the modified conduction band and valence band energies, respectively.

### Note:

These quantities can also be output by using the `Plot` section, but the values are not accurate since the resolution of the original mesh might be much lower than the nonlocal mesh. Therefore, always use `NonLocalPlot` to plot `BarrierLowering`-related quantities.

## Schrödinger Equation-Based Tunneling Probability

In addition to the WKB-based models discussed in [WKB Tunneling Probability on page 833](#), Sentaurus Device can compute tunneling probabilities based on the Schrödinger equation. For the Schrödinger equation-based model, Sentaurus Device computes the tunneling probability  $\Gamma_{CC,v}(E)$  (or  $\Gamma_{VV,v}(E)$ ) by solving the 1D Schrödinger equation:

$$-\frac{d}{dr} \frac{\hbar^2}{m(r)} \frac{d}{dr} \Psi(r) + E_{C,v}(r) \Psi(r) = E(r) \Psi(r) \quad (826)$$

between the outermost classical turning points that belong to the tunneling energy  $E$ . For  $m(r)$ , Sentaurus Device uses the tunneling mass `mt` (see [Nonlocal Tunneling Parameters on page 830](#)).  $E_{C,v}$  is the conduction-band energy shifted by the  $v$ -th value in `eoffset` (see [Nonlocal Tunneling Parameters on page 830](#)).

For boundary conditions, Sentaurus Device assumes incident and reflected plane waves outside the barrier on one side, and an evanescent plane wave on the other side. The

## Chapter 24: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

energy for the plane waves is the greater of  $kT_n$  and  $E - E_{C,V}$ , where  $T_n$  and  $E_{C,V}$  are taken at the point immediately outside the barrier on the respective side. The masses outside the barrier are the tunneling masses  $m_t$  that are valid there.

### Note:

The option `Schroedinger` does not work with the option `TwoBand` or `Band2Band`.

## Density Gradient Quantization Correction

Two different quantization corrections to tunneling are available with the density gradient model (see [Density Gradient Model on page 362](#)).

The first correction is intended for tunneling from an inversion layer through an insulator. The corrections are multipliers to  $T_{CC,V}(r, \varepsilon)$  and read:

$$\exp \left[ \alpha_n \frac{\Lambda_{fb,n}(r) - \Lambda_n(r)}{kT_n(r)} \right] \quad (827)$$

where  $\alpha_n$  is an adjustable parameter (see [Nonlocal Tunneling Parameters on page 830](#)), and  $\Lambda_{fb,n}$  is the solution of the 1D density gradient equations for flatband (that is,  $\phi = \text{const}$ ) conditions. A multiplier analogous to that in [Equation 827](#) exists for  $T_{VV,V}$ .

The second correction is intended for situations where quantization is due to confinement perpendicular to the tunneling direction, for example, for tunneling in the channel direction of a nanowire. For this correction, for the computation of tunneling,  $\Lambda_n$  and  $-\Lambda_p$  are added to the conduction band edge and the valence band edge, with prefactors as described in [Nonlocal Tunneling Parameters on page 830](#).

The motivation for doing this is: If confinement and tunneling are perpendicular, the problem can be separated into a 1D tunneling problem and 2D quantization problems. From the 2D quantization problems, a subband profile along the tunneling direction can be extracted. The lowest energy subbands for electrons and holes form the effective barrier for the 1D tunneling problem. The lowest subband energy profile is approximated by the sum of the band edge and the quantum potential. Note that the latter approximation interprets the quantum potential as an energy-like quantity, ignoring the fact that the quantum potential also describes the shape of the wavefunction.

## Multivalley Band Structure and Geometric Quantization

When used with the `Multivalley` option, the tunneling model uses the multivalley band structure as described in [Multivalley Band Structure on page 327](#). In particular, this means:

- Multiple tunneling processes are accounted for. For band-to-band tunneling, tunneling between all valleys of the conduction band to all valleys of the valence band is accounted for. For tunneling within the conduction band or within the valence band, tunneling within each valley is accounted for; while no tunneling between different valleys is taken into account.

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

- The tunneling mass is the inverse of the reciprocal mass tensor specified by the multivalley parameters, projected in the tunneling direction. Where no multivalley parameters are available, the tunneling mass is determined from the tunneling parameters as described in [Nonlocal Tunneling Parameters on page 830](#).
- The conduction and valence band edges used to compute tunneling include the band offsets as described by the multivalley model parameters.
- When the multivalley band structure is used with the `ThinLayer` model, the resulting quantum corrections due to geometric confinement are used in the tunneling model as well. The details of how this is performed are described next.

For the interplay of tunneling and quantization, two limiting cases can be distinguished:

- When tunneling is perpendicular to the confinement, the quantization increases the tunneling barrier, as discussed in [Density Gradient Quantization Correction on page 837](#).
- When tunneling is in the confinement direction, quantization does not affect the tunneling barrier. However, quantization affects the density-of-states available to carriers in the bands to which they tunnel.

To connect the two limiting cases, Sentaurus Device uses the following heuristic approach: For each point along the tunneling path and for each valley, the angle  $\beta$  between the tunneling direction and the confinement direction is computed in a coordinate system where the mass tensor of the valley becomes diagonal. From  $\beta$  and the total quantization energy  $\varepsilon_1^i$  given by [Equation 202](#), a “perpendicular” quantization energy is obtained,  $\varepsilon_{\perp} = (1 + -\cos^2\beta)\varepsilon_1^i$ . The latter is added to the band edges in computing the tunneling probability (see [WKB Tunneling Probability on page 833](#) and [Schrödinger Equation-Based Tunneling Probability on page 836](#)).

The total energy is added to the bands in the expressions for the energy and position integrations (see [Nonlocal Tunneling Current on page 838](#)). None of these terms is added for the expressions for carrier heating (see [Carrier Heating on page 841](#)).

#### Note:

It is recommended that the number of valleys and their names are the same on an entire tunneling path, even when the path runs in more than one region.

## Nonlocal Tunneling Current

For a point at  $u$ , the expression for the net conduction band electron recombination rate due to tunneling to and from the  $v$ -th shifted conduction band at point  $l < u$  with energy  $\varepsilon$  is:

$$R_{CC,v}(u, l, \varepsilon) - G_{CC,v}(u, l, \varepsilon) = \frac{A_{CC}}{qk} \vartheta \left[ \varepsilon - E_{C,v}(u), -\frac{dE_{C,v}}{du}(u) \right] \vartheta \left[ \varepsilon - E_{C,v}(l), \frac{dE_{C,v}}{dl}(l) \right] \Gamma_{CC,v}(u, l, \varepsilon) \\ \times \left[ T_n(u) \ln \frac{1}{1 + \exp \left[ \frac{E_{F,n}(u) - \varepsilon}{kT_n(u)} \right]} - T_n(l) \ln \frac{1}{1 + \exp \left[ \frac{E_{F,n}(l) - \varepsilon}{kT_n(l)} \right]} \right] \quad (828)$$

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

where  $\vartheta(x, y) = \delta(x)|y|\Theta(y)$ ,  $A_{CC} = g_C A_0$  is the effective Richardson constant (with  $A_0 = 4\pi m_0 kq/h^3$  the Richardson constant for free electrons), and  $g_C$  is a fit parameter (see [Nonlocal Tunneling Parameters on page 830](#)).  $R_{CC, v}$  relates to the first term and  $G_{CC, v}$  to the second term in the second line of [Equation 828](#).  $\Gamma_{CC, v}$  is the tunneling probability (see [Equation 821](#)).

For nonlocal lines with zero `Permeation`, the second  $\vartheta$ -function is replaced with  $\Theta[\epsilon - E_{C, v}(l)]\delta(l - 0^+)$  (with  $0^+$  infinitesimally smaller than 0). For contacts, metals, or in the presence of the option `BandGap`, the second  $\vartheta$ -function is replaced with  $\delta(l - 0^+)$ .

The contribution to the electron tunneling current density by electrons that tunnel from the conduction band at points above  $l$  to the conduction band at point  $l$  is the integral over the recombination rate ([Equation 828](#)):

$$\frac{dJ_{CC}}{dl}(l) = -q \int_{v=0}^{\infty} \int_{l=-\infty}^{\infty} [R_{CC, v}(u, l, \epsilon) - G_{CC, v}(u, l, \epsilon)] d\epsilon du \quad (829)$$

The options for `hBarrierTunneling` and the expressions for the valence band to valence band tunneling current density  $j_{VV}$  are analogous.

Tunneling current calculation is very sensitive to mesh density. To reduce mesh sensitivity and to achieve reasonable results with a relatively coarse mesh, a phenomenological correction function is used to fine-tune the Fermi potential and the integration weight along the nonlocal line:

$$f(l) = C \exp(-\lambda l) \sin(\omega l + \theta\pi) \min(1, (\alpha_0 \Delta l)^{\beta_0}) \min(1, (\alpha_1 \Delta l_{\perp})^{\beta_1}) \min(1, \frac{N}{N_0})^{\beta_0 + \beta_1} \quad (830)$$

where:

- $C$  is the coefficient of the correction function.
- $\lambda$  is the damping ratio, which controls how fast the correction function approaches 0.
- $\omega$  is the periodicity of the correction function.
- $\theta$  is the phase of the correction function.
- $\alpha$  and  $\beta$  control the damping with regard to mesh density.
- $N_0$  is a parameter that controls doping concentration dependency.

By default,  $C=0$  so that the correction function has no impact.  $\Delta l$  is the distance from the nonlocal line base to its first real mesh point along the line, and  $\Delta l_{\perp}$  is the distance from the nonlocal line base to its neighbor mesh point perpendicular to the line direction. When the real mesh is sufficiently fine,  $\Delta l$  is very small, so that the correction function becomes negligible.

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

These parameters are specified in the `BarrierTunneling` section of the parameter file:

```
Electrode = "source" {
    BarrierTunneling {
        C = 0.12 , 0.12
        lambda = 1e7 , 1e7
        omega = 5e7 , 5e7
        phase= 0.0 , 0.0
    }
}
```

*Table 145 Model parameters for correction function*

Symbol	Parameter name	Default value (electrons)	Unit
$C$	<code>C</code>	0.0	1
$\lambda$	<code>lambda</code>	$1 \times 10^7$	1/cm
$\omega$	<code>omega</code>	$5 \times 10^7$	1/cm
$\theta$	<code>phase</code>	0.0	1
$\alpha_0$	<code>alpha0</code>	$5 \times 10^7$	1/cm
$\beta_0$	<code>beta0</code>	3.0	1
$\alpha_1$	<code>alpha1</code>	$5 \times 10^7$	1/cm
$\beta_1$	<code>beta1</code>	0.0	1
$N_0$	<code>N0</code>	$1 \times 10^{20}$	cm <sup>-3</sup>

It has been noted that, in a typical transistor, the correction added to the source contact has a more pronounced impact than the drain contact.

## Band-to-Band Contributions to Nonlocal Tunneling Current

When you specify the `Band2Band` option to `eBarrierTunneling` (see [Table 242 on page 1634](#)), the total current that flows to the conduction band at point  $l$  also contains electrons that originate from the valence band at position  $u > l$ .

## Chapter 24: Tunneling

### Nonlocal Tunneling at Interfaces, Contacts, and Junctions

For `Band2Band=Simple`, the recombination rate of the valence-band electrons with energy  $\varepsilon$  at point  $u$  (in other words, the generation rate of holes at  $u$ ) due to tunneling to or from the  $v$ -th shifted conduction band at  $l$  is:

$$R_{CV,v}(u, l, \varepsilon) - G_{CV,v}(u, l, \varepsilon) = \frac{A_{CV}}{2qk} \vartheta \left[ \varepsilon - E_{V,v}(u), \frac{dE_{V,v}(u)}{du} \right] \vartheta \left[ \varepsilon - E_{C,v}(l), \frac{dE_{C,v}(l)}{dl} \right] \Gamma_{CV,v}(u, l, \varepsilon) \\ \times [T_p(u) + T_n(l)] \left[ \frac{1 + \exp \left[ \frac{\varepsilon - E_{F,p}(u)}{kT_p(u)} \right]^{-1}}{1 + \exp \left[ \frac{\varepsilon - E_{F,n}(l)}{kT_n(l)} \right]^{-1}} \right] \quad (831)$$

where  $A_{CV} = \sqrt{g_C g_V} A_0$ . The prefactor  $g_V$  is a fit parameter, see [Nonlocal Tunneling Parameters on page 830](#).  $\Gamma_{CV,v}$  is the band-to-band tunneling probability discussed above. The modifications for zero `Permeation`, contacts, metals, and the `BandGap` option are as for [Equation 828](#).

The current density of electrons that tunnel from the valence band at points above  $l$  to the conduction band at point  $l$  is the integral over the recombination rate ([Equation 831](#)):

$$\frac{dj_{CV}}{dl}(l) = -q \sum_{v=1}^{\infty} \int_{-\infty}^{\infty} [R_{CV,v}(r, l, \varepsilon) - G_{CV,v}(r, l, \varepsilon)] d\varepsilon dr \quad (832)$$

For band-to-band tunneling processes, the energy of the tunneling particles often lies deep in the gap of the barrier. The single-band dispersion relations that Sentaurus Device uses by default (see [Equation 819](#) and [Equation 820](#)) are based on the band structure near the band edges, and might not be useful in this regime. The two-band dispersion relation according [Equation 823](#) is a better choice. To use the two-band dispersion relation, specify the option `TwoBand` to `eBarrierTunneling` (see [Table 242 on page 1634](#)).

The options for `hBarrierTunneling` and the expressions for the conduction band to valence band tunneling current density  $j_{VC}$  are analogous.

## Carrier Heating

In hydrodynamic simulations, carrier transport leads to energy transport and, therefore, to heating or cooling of electrons and holes. The energy transport has a convective and a Peltier part. By default, Sentaurus Device ignores the Peltier part. To include the Peltier terms for the tunneling particles, specify the option `PeltierHeat` to `eBarrierTunneling` or `hBarrierTunneling` (see [Table 242 on page 1634](#)).

The convective part of the heat generation in the conduction and valence bands at position  $u$  due to tunneling of carriers with energy  $\varepsilon$  to and from the  $v$ -th shifted conduction band at  $l < u$  is approximated as:

$$H_{conv,CC,v}(u, l, \varepsilon) = \frac{\delta}{2} [G_{CC,v}(u)kT_n(l) - R_{CC,v}(u)kT_n(u)] \quad (833)$$

## Chapter 24: Tunneling

### References

and:

$$H_{\text{conv},\text{CV},v}(u, l, \varepsilon) = \frac{\delta}{2} [R_{\text{CV},v}(u)kT_p(u) - G_{\text{CV},v}(u)kT_n(l)] \quad (834)$$

By default, Sentaurus Device neglects Peltier heating and uses  $\delta = 3$ , which corresponds to the three degrees of freedom of the carriers. If Peltier heating is included in a simulation, the convective contribution due to one degree of freedom is already contained in the Peltier heating term. Therefore, in this case, Sentaurus Device uses  $\delta = 2$ . The convective parts of the heat generation in the conduction band at  $l$ , due to tunneling of carriers of energy  $\varepsilon$  from the conduction band and the valence band at  $u > l$ , are  $-H_{\text{conv},\text{CC},v}(u, l, \varepsilon)$  and  $-H_{\text{conv},\text{CV},v}(u, l, \varepsilon)$ .

The expressions for the convective part of the heat generation in the valence band are analogous.

If the computation of Peltier heating is activated (see [Table 242 on page 1634](#)), Sentaurus Device computes additional heating terms. The Peltier part of the heat generation in the electron system at point  $u$  due to tunneling to and from the  $v$ -th shifted conduction band at point  $l < u$  is:

$$H_{\text{Pelt},\text{CC},v}(u, l, \varepsilon) = [E_C(u^+) - \varepsilon] [R_{\text{CC},v}(u, l, \varepsilon) - G_{\text{CC},v}(u, l, \varepsilon)] \quad (835)$$

where  $u^+$  is infinitesimally larger than  $u$ . [Equation 835](#) vanishes everywhere except at abrupt jumps of the band edge. The Peltier part of the heat generation in the electron system at point  $l$  due to tunneling from the conduction band at  $u > l$  obeys an equation similar to [Equation 835](#), with  $E_C(u^+) - \varepsilon$  replaced by  $\varepsilon - E_C(l^-)$ .

When the Band2Band option is used (see [Table 242 on page 1634](#)), Sentaurus Device also takes into account the band-to-band terms of the Peltier part of the heat generation at point  $u$ :

$$H_{\text{Pelt},\text{CV},v}(u, l, \varepsilon) = [E_V(u^+) - \varepsilon] [R_{\text{CV},v}(u, l, \varepsilon) - G_{\text{CV},v}(u, l, \varepsilon)] \quad (836)$$

and, similarly, for the Peltier heat generation at point  $l$ .

The expressions for  $H_{\text{Pelt},\text{VV},v}$  and  $H_{\text{Pelt},\text{VC},v}$  are analogous to those for conduction band tunneling.

---

## References

- [1] A. Schenk and G. Heiser, "Modeling and simulation of tunneling through ultra-thin gate dielectrics," *Journal of Applied Physics*, vol. 81, no. 12, pp. 7900–7908, 1997.
- [2] M. Ieong *et al.*, "Comparison of Raised and Schottky Source/Drain MOSFETs Using a Novel Tunneling Contact Model," in *IEDM Technical Digest*, San Francisco, CA, USA, pp. 733–736, December 1998.

## Chapter 24: Tunneling

### References

- [3] F. Li *et al.*, "Compact Model of MOSFET Electron Tunneling Current Through Ultra-thin SiO<sub>2</sub> and High-k Gate Stacks," in *Device Research Conference*, Salt Lake City, UT, USA, pp. 47–48, June 2003.
- [4] L. F. Register, E. Rosenbaum, and K. Yang, "Analytic model for direct tunneling current in polycrystalline silicon-gate metal–oxide–semiconductor devices," *Applied Physics Letters*, vol. 74, no. 3, pp. 457–459, 1999.

# 25

## Hot-Carrier Injection

---

*This chapter discusses the hot-carrier injection models used in Sentaurus Device.*

---

### Overview of Hot-Carrier Injection Models

Hot-carrier injection is a mechanism for gate leakage. The effect is especially important for write operations in EEPROMs. Sentaurus Device provides different built-in hot-carrier injection models and a PMI for user-defined models:

- Classical lucky electron injection (Maxwellian energy distribution)
- Fiegnan's hot-carrier injection (non-Maxwellian energy distribution)
- SHE distribution hot-carrier injection (non-Maxwellian energy distribution calculated from the spherical harmonics expansion (SHE) of the Boltzmann transport equation)
- Hot-carrier injection PMI (see [Hot-Carrier Injection on page 1465](#))

To activate hot-carrier injection models for electrons (or holes), use the options `eLucky` (`hLucky`), `eFiegnan` (`hFiegnan`), `eSHEDistribution` (`hSHEDistribution`), or `PMIModel_name(electron)` (`PMIModel_name(hole)`) to the `GateCurrent` statement in an interface-specific `Physics` section.

To activate the models for both carrier types, use the `Lucky`, `Fiegnan`, `SHEDistribution`, or `PMIModel_name( )` options. Hot-carrier injection models can be combined with all tunneling models (see [Chapter 24 on page 819](#)).

**Note:**

The SHE distribution hot-carrier injection model calculates the tunneling component together with the thermionic emission term. Therefore, combining it with other tunneling models can result in double-counting of the tunneling component.

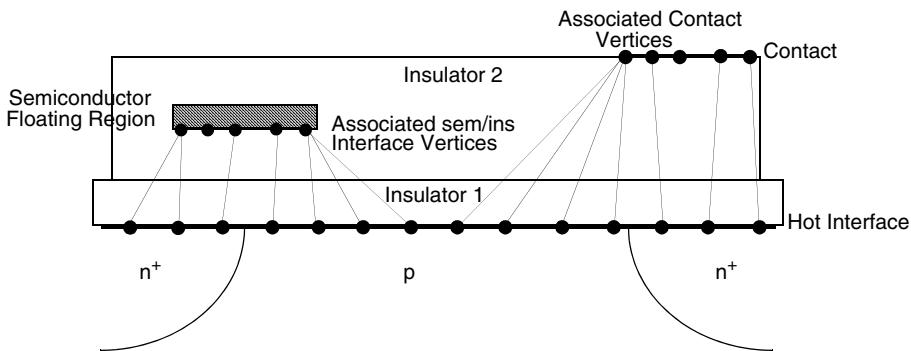
The meaning of a specification in the global `Physics` section is the same as for the Fowler–Nordheim model (see [Fowler–Nordheim Tunneling on page 820](#)).

## Destination of Injected Current

The destination of the injection current depends on the user selection and the material properties of the hot interface (interface source for hot carriers).

When the hot interface is a semiconductor–insulator interface, the injection current is sent nonlocally across the insulator region to an associated closest vertex. For each vertex of a hot interface, Sentaurus Device searches for an associated closest vertex located on a contact or semiconductor–insulator interface. The contacts or semiconductor–insulator interfaces used in the searching algorithm must be connected through adjacent insulator regions to the hot interface. Then, each vertex of the hot interface is associated with the closest vertex on a valid contact or semiconductor–insulator interface. Each vertex of the hot interface has either an associated contact vertex or an associated semiconductor–insulator interface vertex (see [Figure 52](#)).

*Figure 52      Mapping of hot interface vertices to associated contact or semiconductor–insulator interface vertices*



When the hot interface is an interface between a semiconductor and a wide-bandgap semiconductor, you can select the destination. By default, the wide-bandgap semiconductor is treated as an insulator region, and the injection current is sent nonlocally across the region to the associated closest vertex as described for semiconductor–insulator interfaces. By using `Thermionic(HCI)` in the `Physics` section of the hot interface and specifying the injection region destination (the wide-bandgap semiconductor region) using the `InjectionRegion` option in the `GateCurrent` section, the points on the hot interface are made double-points and the injection current is injected locally in the same location where it was produced. The injected current on the wide-bandgap semiconductor side of the hot interface becomes the current boundary condition for the continuity equations solved in the region.

In the case where hot carriers are injected into semiconductor floating regions during transient simulations, the way the charge is added to the floating region is determined by the existence of a charge boundary condition (a region with a charge contact) associated with the region. If the charge boundary condition has been specified for a semiconductor floating region, the charge is added as a total charge update, using the integral boundary condition

## Chapter 25: Hot-Carrier Injection

### Overview of Hot-Carrier Injection Models

as defined by [Equation 129 on page 281](#). If the charge boundary condition is not specified for the semiconductor floating region, the charge is added as an interface boundary condition for continuity equations (see [Carrier Injection With Explicitly Evaluated Boundary Conditions for Continuity Equations on page 867](#)). The total hot-carrier injection current added to semiconductor floating regions where the charge boundary condition is specified will be displayed on the electrode associated with the region (floating contact).

Floating semiconductor regions with a charge boundary condition inside a wide-bandgap semiconductor region are made possible by generalizing the concept of a semiconductor floating well.

Sentaurus Device defines a *semiconductor floating well* as a continuous zone of the same doping semiconductor regions in contact with each other and marked in the command file by a charge contact connected to one of regions in the zone. The concept is generalized by treating the inner semiconductor region (embedded floating region) as a separate well. Geometrically, the charge contact is defined on the interface between the inner and outer semiconductor regions. You must indicate in the `Electrode` section of the charge contact using either the `Region` or `Material` keyword which region is to be used as the floating region with a charge boundary condition.

Equilibrium boundary conditions are imposed on the surface of the special floating region previously described. The interface between the inner and outer semiconductor regions is treated as a heterointerface. You must specify the keyword `Heterointerface` in the `Physics` section of the interface between the inner and outer regions. If the inner region is the floating region, the boundary condition for continuity equations at the interface between the two regions, on the outer-region side, will be equilibrium for carrier concentrations. The charge update for the inner floating region is computed as the total current flowing through the floating region boundary (integral of current density over the floating region surface) multiplied by the time step.

Equilibrium carrier concentrations in a point on the double-point interface, on the outer-region side, are computed based on the carrier concentration on the inner-region side of the interface adjusted by  $(N_{C,V}^{WB}/N_{C,V}^{FG})\exp(-\delta E_{C,V}/kT)$  where:

- $\delta E_{C,V}$  is the jump in the conduction band for electrons or in the valence band for holes.
- $N_{C,V}^{WB}/N_{C,V}^{FG}$  is the ratio between density-of-states in the two regions for electrons and holes, respectively.

For example, to have a `PolySi` nanocrystal floating region with a charge boundary condition embedded in an outer `OxideAsSemiconductor` region, with the charge contact "`fg1`" geometrically somewhere on the interface between the two regions, you must specify:

```
Electrode {
    ...
    {Name "fg1" Charge= -1e-18 Material="PolySi" }
    ...
}
```

## Chapter 25: Hot-Carrier Injection

### Overview of Hot-Carrier Injection Models

```
Physics(MaterialInterface="OxideAsSemiconductor/PolySi") {
    Heterointerface                         # required by GeneralizedFG
}
```

Metal floating gates inside a wide-bandgap semiconductor region are allowed as well. In this case, equilibrium boundary conditions are imposed for the carrier densities at the interface between the semiconductor and the metal floating gate (on metal floating contact).

Electrostatic potential at the metal floating contact is determined by the charge on the metal floating gate according to [Equation 127 on page 280](#). The equilibrium quasi-Fermi potential on the metal floating contact is given by  $\Phi = \Phi_n = \Phi_p = \phi + \phi_{MS}$ , where  $\phi_{MS}$  is the workfunction difference between the metal and the semiconductor. Carrier densities at the metal floating contact are determined then by  $\phi_{MS}$ .

The charge update for the metal floating gate is computed as the total current flowing through the metal floating region contact multiplied by the time step.

To activate this special case of the metal floating gate, you specify the `Metal` keyword in the `Electrode` section of the metal floating contact and give the value of the workfunction (which determines  $\phi_{MS}$ ):

```
Electrode {
    ...
    {Name "fg1" Charge=0 Metal Workfunction=4.1}
    ...
}
```

The workfunction can be used as a calibration parameter.

For simple one-gate devices where the sole purpose is to evaluate the hot-carrier current injected across the oxide layer into a semiconductor region, the keyword `GateName` must be specified in the `GateCurrent` statement. In this case, the hot-carrier current is displayed on the electrode specified by `GateName` for both quasistationary and transient simulations.

---

## Injection Barrier and Image Potential

All hot-carrier injection models are implemented as a postprocessing computation after each Sentaurus Device simulation point. The lucky electron model and Fiegna model specify some properties of semiconductor–insulator interfaces. The most important parameter is the height of the Si– $\text{SiO}_2$  barrier ( $E_B$ ).

The height is a function of the insulator field  $F_{\text{ins}}$  and, at any point along the interface, it can be written as:

$$E_B = \begin{cases} E_{B0} - \alpha q |F_{\text{ins}}|^{\frac{1}{2}} - \beta q |F_{\text{ins}}|^{\frac{2}{3}} - P_{\perp} V_{\text{sem}} & F_{\text{ins}} < 0 \\ E_{B0} - \alpha q |F_{\text{ins}}|^{\frac{1}{2}} - \beta q |F_{\text{ins}}|^{\frac{2}{3}} - P_{\perp} V_{\text{sem}} + V_{\text{ins}} & F_{\text{ins}} > 0 \end{cases} \quad (837)$$

## Chapter 25: Hot-Carrier Injection

### Overview of Hot-Carrier Injection Models

where  $E_{B0}$  is the zero field barrier height at the semiconductor–insulator interface. The second term in the equation represents barrier lowering due to the image potential. The third term of the barrier lowering is due to the tunneling processes. For the Si–SiO<sub>2</sub> interface,  $\alpha = 2.59 \times 10^{-4} \text{ V}^{1/2} \text{ cm}^{1/2}$ . There is a large deviation in the literature for the value of  $\beta$ , so it can be considered a fitting parameter. The fourth term  $P_{\perp} V_{\text{sem}}$  appears only in the lucky electron model where  $P_{\perp}$  is a model parameter ( $P_{\perp} = 1$  by default), and  $V_{\text{sem}}$  represents the kinetic energy gain when carriers travel a distance  $y$  to the interface without losing any energy. In the Fiegna model,  $V_{\text{sem}}$  is zero.

The insulator field  $F_{\text{ins}}$  is defined as:

$$F_{\text{ins}} = \vec{F}_{\text{ins}} \cdot \hat{n} \left[ 1 - P_{\text{total}} + P_{\text{total}} \frac{|\vec{F}_{\text{ins}}|}{|\vec{F}_{\text{ins}} \cdot \hat{n}|} \right] \quad (838)$$

where  $\hat{n}$  is a normal vector along the direction of hot-carrier injection, and  $P_{\text{total}}$  is a model parameter ( $P_{\text{total}} = 0$  by default).

In addition, all hot-carrier models contain a probability  $P_{\text{ins}}$  of scattering in the image-force potential well:

$$P_{\text{ins}} = \begin{cases} \stackrel{\circ}{\rightarrow} \exp -\frac{x_0}{\lambda_{\text{ins}}} & F_{\text{ins}} < 0 \\ \stackrel{\circ}{\rightarrow} \exp -\frac{t_{\text{ins}} - x_0}{\lambda_{\text{ins}}} P_{\text{repel}} & F_{\text{ins}} > 0 \end{cases} \quad (839)$$

where  $\lambda_{\text{ins}}$  is the scattering mean free path in the insulator,  $t_{\text{ins}}$  is the distance between the vertex at the hot interface and the closest associated vertex,  $P_{\text{repel}}$  is a model parameter ( $P_{\text{repel}} = 1$  by default), and the distance  $x_0$  is given as:

$$x_0 = \min \sqrt{\frac{q}{16\pi\epsilon_{\text{ins}}|F_{\text{ins}}|}}, \frac{t_{\text{ins}}}{2} \quad (840)$$

In this expression,  $\tilde{\epsilon}_{\text{ins}} = \epsilon_{\text{ins}}(\epsilon_{\text{sem}} + \epsilon_{\text{ins}})/(\epsilon_{\text{sem}} - \epsilon_{\text{ins}})$  is the effective dielectric constant of the insulator where  $\epsilon_{\text{sem}}$  and  $\epsilon_{\text{ins}}$  are the high-frequency dielectric constant of the semiconductor and insulator, respectively. In the lucky electron model and Fiegna model,  $\tilde{\epsilon}_{\text{ins}}$  is directly accessible from the parameter file. In the SHE distribution model,  $\epsilon_{\text{sem}}$  and  $\epsilon_{\text{ins}}$  are separately set from the parameter file.

## Effective Field

The lucky electron model and Fiegna model have an effective electric field  $F_{\text{eff}}$  as a parameter. In Sentaurus Device, there are three possibilities to calculate the effective field:

- With the electric field parallel to the carrier flow (switched on by the keyword `Eparallel`, which is default for hot carrier currents). See [Driving Force on page 509](#).

## Chapter 25: Hot-Carrier Injection

### Classical Lucky Electron Injection

- With recomputation of the carrier temperature of the hydrodynamic simulation (switched on by the keyword `CarrierTempDrive`). See [Avalanche Generation With Hydrodynamic Transport on page 510](#).
- With a simplified approach (compared to the second method): The drift-diffusion model is used for the device simulation, and carrier temperature is estimated as the solution of the simplified and linearized energy balance equation. As this is a postprocessing calculation, the keyword `CarrierTempPost` activates this option.

These keywords are parameters of the model keywords. For example, the lucky electron model looks like `eLucky(CarrierTempDrive)`. However, you must remember that, if the model includes the keyword `CarrierTempDrive`, then `Hydro` and a carrier temperature calculation must be specified in the `Physics` section.

---

## Classical Lucky Electron Injection

The classical total lucky electron current from an interface to a gate contact can be written as [1]:

$$I_g = J_n(x, y) P_s P_{\text{ins}} \int_{E_B}^{\infty} P_\epsilon P_r d\epsilon dx dy \quad (841)$$

where  $P_s$  is the probability that the electron will travel a distance  $y$  to the interface without losing any energy,  $P_\epsilon d\epsilon$  is the probability that the electron has energy between  $\epsilon$  and  $\epsilon + d\epsilon$ ,  $P_{\text{ins}}$  is the probability of scattering in the image force potential well ([Equation 839](#)), and  $P_r$  is the probability that the electron will be redirected. These probabilities are given by the expressions:

$$P_r(\epsilon) = \frac{1}{2\lambda_r} 1 - \sqrt{\frac{E_B}{\epsilon}} \quad (842)$$

$$P_s(y) = \exp -\frac{y}{\lambda} \quad (843)$$

$$P_\epsilon(\epsilon) = \frac{1}{\lambda F_{\text{eff}}} \exp -\frac{\epsilon}{\lambda F_{\text{eff}}} \quad (844)$$

where:

- $\lambda$  is the scattering mean free path in the semiconductor.
- $\lambda_r$  is redirection mean free path.
- $F_{\text{eff}}$  is the effective electric field (see [Effective Field on page 848](#)).
- $E_B$  is the height of the semiconductor–insulator barrier.

## Chapter 25: Hot-Carrier Injection

### Fiegna Hot-Carrier Injection

The model coefficients can be changed in the parameter file in the section:

```
LuckyModel { ... }
```

*Table 146 Coefficients and their default values for the lucky electron model*

Symbol	Parameter name (Electrons)	Default value (Electrons)	Parameter name (Holes)	Default value (Holes)	Unit
$\lambda$	eLsem	$8.9 \times 10^{-7}$	hLsem	$1.0 \times 10^{-7}$	cm
$\lambda_{\text{ins}}$	eLins	$3.2 \times 10^{-7}$	hLins	$3.2 \times 10^{-7}$	cm
$\lambda_r$	eLsemR	$6.2 \times 10^{-6}$	hLsemR	$6.2 \times 10^{-6}$	cm
$E_{B0}$	eBar0	3.1	hBar0	4.7	eV
$\alpha$	eBL12	$2.6 \times 10^{-4}$	hBL12	$2.6 \times 10^{-4}$	$(V \cdot cm)^{1/2}$
$\beta$	eBL23	$3.0 \times 10^{-5}$	hBL23	$3.0 \times 10^{-5}$	$(V \cdot cm^2)^{1/3}$
$\tilde{\epsilon}_{\text{ins}}$	eps_ins	3.1	eps_ins	3.1	1
$P_{\perp}$	Pvertical	1	Pvertical	1	1
$P_{\text{repel}}$	Prepel	1	Prepel	1	1
$P_{\text{total}}$	Ptotal	0	Ptotal	0	1

---

## Fiegna Hot-Carrier Injection

The total hot-carrier injection current according to the Fiegna model [2] can be written as:

$$I_g = q P_{\text{ins}} \int_{E_{B0}}^{\infty} v_{\perp}(\epsilon) f(\epsilon) g(\epsilon) d\epsilon ds \quad (845)$$

where:

- $\epsilon$  is the electron energy.
- $E_{B0}$  is the height of the semiconductor–insulator barrier.
- $v_{\perp}$  is the velocity normal to the interface.
- $f(\epsilon)$  is the electron energy distribution.

## Chapter 25: Hot-Carrier Injection

### Fiegna Hot-Carrier Injection

- $g(\varepsilon)$  is the density-of-states of the electrons.
- $P_{\text{ins}}$  is the probability of scattering in the image force potential well as described by [Equation 839](#).
- $ds$  is an integral along the semiconductor–insulator interface.

The following expression for the electron energy distribution was proposed for a parabolic and an isotropic band structure, and equilibrium between lattice and electrons:

$$f(\varepsilon) = A \exp -\chi \frac{\varepsilon^3}{F_{\text{eff}}^{1.5}} \quad (846)$$

Therefore, the gate current can be rewritten as:

$$I_g = q \frac{A}{3\chi} P_{\text{ins}} n \frac{F_{\text{eff}}^{3/2}}{\sqrt{E_B}} e^{-\frac{\chi E_B^3}{F_{\text{eff}}^{3/2}}} ds \quad (847)$$

where  $F_{\text{eff}}$  is an effective field (see [Effective Field on page 848](#)).

*Table 147 Coefficients and their default values for the Fiegna model*

Symbol	Parameter name (Electrons)	Default value (Electrons)	Parameter name (Holes)	Default value (Holes)	Unit
$A$	eA	$4.87 \times 10^4$	hA	$4.87 \times 10^4$	$\text{cm/s/eV}^{2.5}$
$\chi$	eChi	$1.3 \times 10^8$	hChi	$1.3 \times 10^8$	$(\text{V/cm/eV})^{1.5}$
$\lambda_{\text{ins}}$	eLins	$3.2 \times 10^{-7}$	hLins	$3.2 \times 10^{-7}$	cm
$E_{B0}$	eBar0	3.1	hBar0	4.7	eV
$\alpha$	eBL12	$2.6 \times 10^{-4}$	hBL12	$2.6 \times 10^{-4}$	$(\text{V} \cdot \text{cm})^{1/2}$
$\beta$	eBL23	$1.5 \times 10^{-5}$	hBL23	$1.5 \times 10^{-5}$	$(\text{V} \cdot \text{cm}^2)^{1/3}$
$\tilde{\epsilon}_{\text{ins}}$	eps_ins	3.1	eps_ins	3.1	1
$P_{\text{repel}}$	Prepel	1	Prepel	1	1
$P_{\text{total}}$	Ptotal	0	Ptotal	0	1

These coefficients can be changed in the parameter file in the `Fiegnamode1` section.

## SHE Distribution Hot-Carrier Injection

To obtain the hot-carrier injection current, accurate knowledge of the nonequilibrium electron-energy distribution is required. The spherical harmonics expansion (SHE) distribution hot-carrier injection model calculates the hot-carrier injection current using the nonequilibrium energy distribution  $f$  obtained from the lowest-order SHE of the semiclassical Boltzmann transport equation (BTE) (see [Spherical Harmonics Expansion Method on page 853](#)).

The total hot-carrier injection current is obtained from [3]:

$$I_g = \frac{2qA g_v}{4} P_{\text{ins}} \left[ \int_0^{\infty} g(\varepsilon) v(\varepsilon) f(\varepsilon) \int_0^1 \Gamma \left[ \varepsilon - \frac{\hbar^3 g(\varepsilon) v(\varepsilon) x}{8\pi m_{\text{ins}}} \right] dx \, d\varepsilon \right] ds \quad (848)$$

where:

- $A$  is a dimensionless prefactor ( $A = 1$  by default).
- $g_v$  is the valley degeneracy factor.
- $P_{\text{ins}}$  is the probability of electrons moving from the interface to the barrier peak without scattering (see [Equation 839 on page 848](#)).
- $g$  is the density-of-states per valley and per spin.
- $v$  is the magnitude of the electron velocity.
- $\Gamma$  is the transmission coefficient obtained from the WKB approximation including the image-potential barrier-lowering.
- $m_{\text{ins}}$  is the insulator effective mass.
- $ds$  is an integral along the semiconductor–insulator interface.

The transmission coefficient can be written as:

$$\Gamma(\varepsilon_{\perp}) = \exp \left( -\frac{2}{\hbar} \int_0^{t_{\text{ins}}} \sqrt{2m_{\text{ins}}[E_B(r) - \varepsilon_{\perp}]} \Theta[E_B(r) - \varepsilon_{\perp}] dr \right) \quad (849)$$

$$E_B(r) = E_{B0} + qF_{\text{ins}}r + E_{\text{im}}(r) \quad (850)$$

$$E_{\text{im}}(r) = -\frac{q^2}{16\pi\varepsilon_{\text{ins}}} \frac{\varepsilon_{\text{sem}} - \varepsilon_{\text{ins}}}{\varepsilon_{\text{sem}} + \varepsilon_{\text{ins}}} \sum_{n=0}^{\infty} \left[ \frac{1}{nt_{\text{ins}} + r} + \frac{1}{(n+1)t_{\text{ins}} - r} - \frac{2}{(n+1)t_{\text{ins}}} \frac{\varepsilon_{\text{sem}} - \varepsilon_{\text{ins}}}{\varepsilon_{\text{sem}} + \varepsilon_{\text{ins}}} \right] \quad (851)$$

where  $E_{B0}$  is the barrier height.

To use the SHE distribution hot-carrier injection model, you must obtain the distribution function by solving the SHE method (see [Spherical Harmonics Expansion Method on page 853](#)). [Equation 851](#) represents the image-potential barrier-lowering. You can switch off the image-potential barrier-lowering using:

```
Physics(MaterialInterface="Silicon/Oxide") { ...
    GateCurrent( eSHEDistribution( -ImagePotential ) )
}
```

## Spherical Harmonics Expansion Method

The spherical harmonics expansion (SHE) method computes the microscopic carrier energy distribution function  $f(r, \varepsilon)$  by solving the lowest-order SHE of the Boltzmann transport equation (BTE) [4]:

$$-\nabla \cdot \left[ \frac{v^2(\vec{r}, \varepsilon_t)}{3} \tau(\vec{r}, \varepsilon_t) g(\vec{r}, \varepsilon_t) \nabla f(\vec{r}, \varepsilon_t) \right] = g(\vec{r}, \varepsilon_t) s(\vec{r}, \varepsilon_t) \quad (852)$$

where:

- $f$  is the occupation probability of electrons ( $f$  can be larger than one as nondegenerate statistics is assumed).
- $\varepsilon_t$  is the total energy including the conduction band energy  $E_C$  and the kinetic energy  $\varepsilon$ .
- $v$  is the magnitude of the electron velocity.
- $1/\tau$  is the total scattering rate.
- $g$  is the density-of-states per valley and per spin.
- $s$  is the net in-scattering rate due to inelastic scattering and generation–recombination processes.

In [Equation 852](#),  $g(\varepsilon)$  and  $v(\varepsilon)$  can be obtained from the energy–wavevector dispersion relation,  $\varepsilon_b(\vec{k})$ , of semiconductors:

$$g_v g(\varepsilon) = g_v \quad g_b(\varepsilon) = \sum_{b=1}^{N_b} \frac{1}{4\pi^2 h v_b(\vec{k})} d\vec{s}_{\vec{k}} \quad (853)$$

$$g_v g(\varepsilon) v^2(\varepsilon) = g_v \quad g_b(\varepsilon) v_b^2(\varepsilon) = \sum_{b=1}^{N_b} \frac{v_b(\vec{k})}{4\pi^2 h} d\vec{s}_{\vec{k}} \quad (854)$$

where:

- $\varepsilon = \varepsilon_t - E_C(r)$  is the kinetic energy.
- $g_v$  is the valley degeneracy.
- $b$  is the band index.
- $N_b$  is the number of bands.
- $v_b(\vec{k})$  is the magnitude of wavevector-dependent group velocity.
- $h$  is Planck's constant.
- The integration is over the equienergy surface of the first Brillouin zone.

In addition, the energy-dependent square wavevector is defined as:

$$g_v k^2(\varepsilon) = g_v \sum_{b=1}^{N_b} k_b^2(\varepsilon) = \frac{1}{4\pi} \sum_{b=1}^{N_b} ds_{\vec{k}} \quad (855)$$

These energy-dependent, band structure-related quantities can be obtained either from the precalculated band-structure file or from the single band, analytic, nonparabolic, band-structure model. For more information on the band-structure file, see [Using Spherical Harmonics Expansion Method on page 857](#).

The analytic nonparabolic band-structure model gives [5]:

$$\frac{v^2(\varepsilon)}{3} = \frac{2\varepsilon(1+\alpha\varepsilon)}{3m_c(1+2\alpha\varepsilon)^2} \quad (856)$$

$$g(\varepsilon) = \frac{2\pi(2m_n)^{3/2}}{h^3} [\varepsilon(1+\alpha\varepsilon)]^{1/2} (1+2\alpha\varepsilon) \quad (857)$$

$$k^2(\varepsilon) = \pi h g(\varepsilon) v(\varepsilon) \quad (858)$$

where  $\alpha$  is the nonparabolicity factor,  $m_c$  is the conductivity effective mass, and  $m_n$  is the density-of-states effective mass.

The total scattering rate and the net in-scattering rate can be written as:

$$\frac{1}{\tau(\varepsilon)} = \frac{1}{\tau_c(\varepsilon)} + \frac{1}{\tau_{ii}(\varepsilon)} + \frac{1}{\tau_{ac}(\varepsilon)} + \frac{1}{\tau_{ope}(\varepsilon)} + \frac{1}{\tau_{opa}(\varepsilon)} \quad (859)$$

$$s(\varepsilon) = \frac{f_{loc}(\varepsilon) - f(\varepsilon)}{\tau_{ii}(\varepsilon)} + \frac{f(\varepsilon - \varepsilon_{op}) \exp \left[ \frac{\varepsilon_{op}}{kT} \right] - f(\varepsilon)}{\tau_{ope}(\varepsilon)} + \frac{f(\varepsilon + \varepsilon_{op}) \exp \left[ \frac{\varepsilon_{op}}{kT} \right] - f(\varepsilon)}{\tau_{opa}(\varepsilon)} - \frac{R_{net} f_{loc}(\varepsilon)}{n} \quad (860)$$

where:

- $1/\tau_c$  is the Coulomb scattering rate.
- $1/\tau_{ii}$  is the impact ionization scattering rate.
- $1/\tau_{ac}$  is the acoustic phonon scattering rate.
- $1/\tau_{ope}$  is the scattering rate due to optical phonon emissions.
- $1/\tau_{opa}$  is the scattering rate due to optical phonon absorptions.
- $\epsilon_{op}$  is the optical phonon energy.
- $R_{net}$  is the net recombination rate.
- $n$  is the electron density.
- $f_{loc} = \exp[(E_{F,n} - E_C - \epsilon)/kT]$  is the local equilibrium distribution function.

The Coulomb scattering rate can be written as [6]:

$$\frac{1}{\tau_c(\epsilon)} = \frac{q^4 \pi^2 g(\epsilon) N_{i,eff}}{2h\epsilon_{sem}^2 [k^2(\epsilon) + k_0^2]^2} \left[ \ln(1+b) - \frac{b}{1+b} \right] \quad (861)$$

$$b(\epsilon) = \frac{4[k^2(\epsilon) + k_0^2]kT\epsilon_{sem}}{q^2(n+p)} \quad (862)$$

$$N_{i,eff} = \begin{cases} \overset{\circ}{(N_D + N_A)\zeta_{major}(N_{major})} & N_{major} > N_{minor} \\ \overset{\circ}{(N_D + N_A)\zeta_{minor}(N_{minor})} & N_{major} < N_{minor} \end{cases} \quad (863)$$

where:

- $\epsilon_{sem}$  is the dielectric constant of a semiconductor material.
- $k_0^2$  is an adjustable parameter that controls the energy dependency of the impurity scattering.
- $N_{major}$  is  $N_D$  for electrons and  $N_A$  for holes.
- $N_{minor}$  is  $N_A$  for electrons and  $N_D$  for holes.
- $\zeta_{major}$  and  $\zeta_{minor}$  are tabulated fitting functions introduced to match the experimental low-field mobility curve as a function of majority and minority doping concentrations, respectively.

## Chapter 25: Hot-Carrier Injection

### SHE Distribution Hot-Carrier Injection

Two different expressions for the impact ionization scattering rate are available. The first expression can be written as [6]:

$$\frac{1}{\tau_{ii}(\epsilon)} = \begin{cases} \frac{\epsilon - \epsilon_{ii,1}}{1 \text{ eV}}^{v_{ii,1}} s_{ii,1} & \epsilon_{ii,1} < \epsilon < \epsilon_{ii,3} \\ \frac{\epsilon - \epsilon_{ii,2}}{1 \text{ eV}}^{v_{ii,2}} s_{ii,2} & \epsilon > \epsilon_{ii,3} \end{cases} \quad (864)$$

where:

- $s_{ii,1}$  and  $s_{ii,2}$  are the impact ionization coefficients.
- $v_{ii,1}$  and  $v_{ii,2}$  are the exponents.
- $\epsilon_{ii,1}$ ,  $\epsilon_{ii,2}$ , and  $\epsilon_{ii,3}$  are the reference energies.

The second expression can be written as [7]:

$$\frac{1}{\tau_{ii}(\epsilon)} = \sum_{j=1}^3 \frac{\epsilon - \epsilon_{ii,j}}{1 \text{ eV}}^{v_{ii,j}} \Theta(\epsilon - \epsilon_{ii,j}) s_{ii,j} \quad (865)$$

Specifying `ii_formula=1` in the `SHEDistribution` parameter set activates the first expression; while `ii_formula=2` activates the second expression. The impact ionization model parameters for electrons and holes are obtained from [6] and [8], respectively.

The acoustic-phonon and optical-phonon scattering rates can be written as [5][6]:

$$\frac{g(\epsilon)}{\tau_{ac}(\epsilon)} = \sum_{i=1}^{N_b} \sum_{j=1}^{N_b} \frac{4\pi^2 k T D_{ac,ij}^2}{h\rho c_L^2} g_i(\epsilon) g_j(\epsilon) \quad (866)$$

$$\frac{g(\epsilon)}{\tau_{ope}(\epsilon)} = \sum_{i=1}^{N_b} \sum_{j=1}^{N_b} \frac{h D_{op,ij}^2}{2\rho \epsilon_{op}} (N_{op} + 1) g_i(\epsilon) g_j(\epsilon - \epsilon_{op}) \quad (867)$$

$$\frac{g(\epsilon)}{\tau_{opa}(\epsilon)} = \sum_{i=1}^{N_b} \sum_{j=1}^{N_b} \frac{h D_{op,ij}^2}{2\rho \epsilon_{op}} N_{op} g_i(\epsilon) g_j(\epsilon + \epsilon_{op}) \quad (868)$$

where:

- $i$  and  $j$  are the band indices.
- $D_{ac,ij}$  and  $D_{op,ij}$  are the deformation potentials for acoustic and g-type optical phonons, respectively ( $D_{ac,ij} = D_{ac,ji}$  and  $D_{op,ij} = D_{op,ji}$ ).
- $\rho$  is the mass density.
- $c_L$  is the sound velocity.
- $N_{op} = [\exp(\epsilon_{op}/kT) - 1]^{-1}$  is the phonon number.

## Chapter 25: Hot-Carrier Injection

### SHE Distribution Hot-Carrier Injection

If  $D_{ac,ij} = D_{ac}$  and  $D_{op,ij} = D_{op}$  regardless of the band indices, [Equation 866](#), [Equation 867](#), and [Equation 868](#) can be simplified to:

$$\frac{1}{\tau_{ac}(\varepsilon)} = \frac{4\pi^2 k T D_{ac}^2}{h \rho c_L^2} g(\varepsilon) \quad (869)$$

$$\frac{1}{\tau_{ope}(\varepsilon)} = \frac{h D_{op}^2}{2 \rho \epsilon_{op}} (N_{op} + 1) g(\varepsilon - \epsilon_{op}) \quad (870)$$

$$\frac{1}{\tau_{opa}(\varepsilon)} = \frac{h D_{op}^2}{2 \rho \epsilon_{op}} N_{op} g(\varepsilon + \epsilon_{op}) \quad (871)$$

Dirichlet boundary condition as  $f(\varepsilon) = \exp[(E_{F,n} - E_C - \varepsilon)/kT]$  is assumed for electrodes.

Boundary conditions for abrupt heterointerfaces are similar to the corresponding boundary conditions for the thermionic emission model. Assume that at the heterointerface between materials 1 and 2, the conduction band edge jump is positive ( $\Delta E_C = E_{C,2} - E_{C,1} > 0$ ). If  $J_{n,2}(\varepsilon)$  and  $J_{n,1}(\varepsilon)$  are the energy-dependent electron current density per spin and per valley entering material 2 and leaving material 1, the interface condition can be written as:

$$J_{n,2}(\varepsilon) = J_{n,1}(\varepsilon) \quad (872)$$

$$J_{n,2}(\varepsilon) = \frac{q}{4} g_2(\varepsilon) v_2(\varepsilon) [f_2(\varepsilon) - f_1(\varepsilon)] \quad (873)$$

where  $g_i(\varepsilon)$ ,  $v_i(\varepsilon)$ , and  $f_i(\varepsilon)$  are the density-of-states, the group velocity, and the occupation probability of material  $i$ , respectively.

All other boundaries are treated with reflective boundary conditions.

## Using Spherical Harmonics Expansion Method

The electron-energy distribution function is calculated from [Equation 852](#) in the semiconductor regions specified by the global, region-specific, or material-specific Physics section:

```
Physics { eSHEDistribution( <arguments> ) ... }
```

By default,  $g(\varepsilon)$ ,  $v(\varepsilon)$ , and  $k^2(\varepsilon)$  are obtained from the analytic band model. These band structure-related quantities also can be obtained from the default electron-band file `eSHEBandSilicon.dat` in the directory `$STROOT/tcad/$RELEASE/lib/sdevice/` `MaterialDB/she` by specifying the argument `FullBand`:

```
Physics { eSHEDistribution( FullBand ... ) ... }
```

The default electron-band file `eSHEBandSilicon.dat` contains the band-structure quantities obtained from the nonlocal empirical pseudopotential method for relaxed silicon. Similarly, there is a default hole-band file `hSHEBandsilicon.dat` in the same directory.

**Note:**

For silicon regions, it is recommended to use the `FullBand` option as the default model parameters are calibrated based on the full band structure. In addition, there is no performance penalty when using the `FullBand` option.

You also can specify your own band file as:

```
Physics { eSHEDistribution( FullBand = "filename" ... ) ... }
```

The band file is a plain text file composed of  $1 + 3N_b$  columns of data where  $N_b$  is the number of bands ( $1 \leq N_b \leq 4$ ). The first column represents the kinetic energy  $\varepsilon$  [eV]. The subsequent columns represent  $g_v g_b(\varepsilon)$  [ $\text{cm}^{-3} \text{eV}^{-1}$ ],  $g_v g_b(\varepsilon) v_b(\varepsilon)$  [ $\text{cm}^{-1} \text{s}^{-2} \text{eV}^{-1}$ ], and  $g_v k_b^2(\varepsilon)$  [ $\text{cm}^{-2}$ ] for band  $b$  ( $1 \leq b \leq N_b$ ). The kinetic energy should start from zero, and the energy spacing between the neighbor rows should be uniform. See `eSHEBandsilicon.dat` for more information.

When `FullBand` is specified for devices containing SiGe, the band-structure quantities for SiGe regions are taken from mole fraction-dependent files in the same directory where the silicon files are located. The band-structure data was obtained from the nonlocal empirical pseudopotential method for relaxed SiGe with mole-fraction values of 0.0, 0.1, 0.2, ..., 1.0:

- Electron files: `eSHEBandSiGex0.0.dat`, `eSHEBandSiGex0.1.dat`, ...
- Hole files: `hSHEBandSiGex0.0.dat`, `hSHEBandSiGex0.1.dat`, ...

Sentaurus Device automatically chooses the appropriate files based on the average x-mole fraction value in each SiGe region. Linear interpolation of band-structure quantities is used for intermediate x-mole fraction values.

**Note:**

The simulation of electrons in SiGe is deactivated.

Model parameters in the `SHEDistribution` parameter set (see [Table 148 on page 862](#)) can be specified with mole-fraction dependency. As with data from the band-structure files, the average x-mole fraction value in each region is used to determine the parameter values.

Sentaurus Device provides a simplified SHE model based on the relaxation time approximation (RTA) for the hot-carrier injection current computation. Although the RTA is a poor approximation, the RTA can remove the energy coupling and reduce simulation time.

You can activate the RTA mode as follows:

```
Physics { eSHEDistribution( RTA ... ) ... }
```

When the RTA mode is selected, the relaxation time is defined as:

$$\frac{1}{\tau_{\text{RTA}}(\varepsilon)} = \frac{v(\varepsilon)}{\lambda_{\text{sem}}} + \frac{1}{\tau_0} \quad (874)$$

## Chapter 25: Hot-Carrier Injection

### SHE Distribution Hot-Carrier Injection

where  $\lambda_{\text{sem}}$  and  $\tau_0$  are adjustable parameters representing a mean free path and relaxation time. In the RTA mode, [Equation 860](#) is replaced by:

$$s(\varepsilon) = \frac{f_{\text{loc}}(\varepsilon) - f(\varepsilon)}{\tau_{\text{RTA}}(\varepsilon)} \quad (875)$$

#### Note:

In the RTA mode, you must specify `SHEIterations=1` together with `SHESOR` in the global `Math` section. The RTA mode must not be used for self-consistent computations as the carrier flux is not conserved.

In the SHE method, the low-field mobility is determined by the microscopic scattering rate:

$$\mu_{\text{low}} = \frac{q \sum_0^{\infty} \tau(\varepsilon) g(\varepsilon) v^2(\varepsilon) \exp -\frac{\varepsilon}{kT} d\varepsilon}{3kT \sum_0^{\infty} g(\varepsilon) \exp -\frac{\varepsilon}{kT} d\varepsilon} \quad (876)$$

The mobility obtained from [Equation 876](#) and that from the macroscopic mobility model specified in the `Physics` section generally differ. For example, [Equation 876](#) overestimates the low-field mobility in the inversion layer of MOSFETs as the scattering rate in [Equation 859](#) does not account for the mobility degradation at interfaces. To resolve this inconsistency, the Coulomb scattering rate is adjusted locally to match the low-filed mobility obtained from the mobility model. This option is activated by default. To switch off this option, specify:

```
Physics { eSHEDistribution( -AdjustImpurityScattering ... ) ... }
```

Similarly, for the hole-energy distribution function, specify `hSHEDistribution` in the `Physics` section.

[Equation 852](#) is a coupled energy-dependent conservation equation with diffusion and source terms. The number of unknown variables in the SHE method is much larger than that in the drift-diffusion model or hydrodynamic model because of the additional total energy coordinate.

By default, the blockwise successive over-relaxation (SOR) method is used to solve the equation iteratively where the SOR iteration is performed over different total energies. The linear solver for the block system, the number of SOR iterations, and the SOR parameter can be accessed by the keywords `SHEMethod`, `SHEIterations`, and `SHESORParameter` in the global `Math` section.

The default values are:

```
Math {
    SHEMethod=super
    SHEIterations=20
    SHESORParameter=1.1
}
```

## Chapter 25: Hot-Carrier Injection

### SHE Distribution Hot-Carrier Injection

Instead of using the blockwise SOR method, you also can solve [Equation 852](#) for different energies simultaneously by switching off the keyword `SHESOR` in the global `Math` section. Although any matrix solver can be used, the iterative linear solver ILS is the only practical option to solve [Equation 852](#) because of the large matrix size.

The ILS default parameters in `set=3` can be used for the SHE method. For example:

```
Math {
    -SHESOR
    SHEMethod=ILS(set = 3)
    ...
}
```

The ILS default parameters in `set=3` are defined as:

```
set (3) {
    iterative (gmres(100), tolrel=1e-8, tolunprec=1e-4, tolabs=0,
               maxit=200);
    preconditioning (ilut(0.00011,-1));
    ordering ( symmetric=rcm, nonsymmetric=mpsilst );
    options ( compact=yes, verbose=0, refinebasis=0,
               refinescaling=none,refineresidual=0 );
}
```

The global `Math` section provides some parameters related to the energy grid specification. The minimum and the maximum of the total energy coordinate are defined as:

$$\varepsilon_{t,\min} = \min[\vec{E}_C(r)] \quad (877)$$

$$\varepsilon_{t,\max} = \max[\vec{E}_C(r)] + \varepsilon_{\text{margin}} \quad (878)$$

where  $\varepsilon_{\text{margin}}$  is an energy margin ( $\varepsilon_{\text{margin}} = 1\text{ eV}$  by default). The energy grid spacing is defined by the fraction of the phonon energy  $\Delta\varepsilon_t = \varepsilon_{\text{op}}/N_{\text{refine}}$  where  $N_{\text{refine}}$  is a positive integer ( $N_{\text{refine}} = 1$  by default).

The parameters  $\varepsilon_{\text{margin}}$  and  $N_{\text{refine}}$  can be set in the global `Math` section:

```
Math {
    SHETopMargin = 1.0 # e_margin [eV]
    SHERefinement = 1   # N_refine
    ...
}
```

While the total energy grid is used for the computation, a uniform kinetic energy grid is used for plotting.

The maximum kinetic energy to be plotted can be specified by the keyword `SHECutoff` (5 eV by default) in the global `Math` section:

```
Math { SHECutoff = 5.0 ... }
```

## Chapter 25: Hot-Carrier Injection

### SHE Distribution Hot-Carrier Injection

By default, the carrier energy distribution is updated in the postprocessing computation after each Sentaurus Device simulation point. You can suppress or activate the postprocessing computation using the `Set` statement of the `Solve` section. For example:

```
Solve { ...
    Set (eSHEDistribution (Frozen)) # freeze the distribution function
    ...
    Set (eSHEDistribution (-Frozen)) # unfreeze the distribution function
    ...
}
```

As long as the distribution function is frozen, the distribution is unchanged during simulation.

Instead of the postprocessing computation, you also can obtain the self-consistent DC solution by using the `Plugin` statement. For example:

```
Solve { ...
    Plugin (iterations=100) { Poisson eSHEDistribution hole }
    ...
}
```

In the self-consistent mode, the carrier density and the terminal current are obtained directly from the SHE method. You also can include the quantum correction in the SHE method. For example:

```
Solve { ...
    Plugin { Coupled {Poisson eQuantumPotential} eSHEDistribution hole }
    ...
}
```

#### Note:

In the self-consistent mode, you must specify the keyword `DirectCurrent` in the global `Math` section. In addition, you might need to increase `SHERefinement` to improve the resolution of the energy grid. The self-consistent mode does not support transient, AC, and noise analysis. In general, the lowest-order SHE method might not be sufficiently accurate to simulate nanoscale transistors as the contribution of higher-order terms increases with decreasing device length [9].

The convergence rate of the `Plugin` method can be very slow when large biases are applied.

For backward compatibility, the following pairs of keywords are recognized as synonyms in the command file:

- `SHEDistribution` and `TailDistribution`
- `eSHEDistribution` and `TaileDistribution`
- `hSHEDistribution` and `TailhDistribution`
- `SHEIterations` and `TailDistributionIterations`

## Chapter 25: Hot-Carrier Injection

### SHE Distribution Hot-Carrier Injection

- `SHEMethod` and `TailDistributionMethod`
- `SHESOR` and `TailDistributionSOR`
- `SHESORParameter` and `TailDistributionSORParameter`

In the PMI, you can read the distribution function, density-of-states, and group velocity obtained from the SHE method using the following read functions:

- `ReadeSHEDistribution`: Returns  $f(\epsilon)$  for electrons.
- `ReadeSHETotalDOS`: Returns  $2g_v g(\epsilon)$  for electrons.
- `ReadeSHETotalGSV`: Returns  $2g_v g(\epsilon)v^2(\epsilon)$  for electrons.
- `ReadhSHEDistribution`: Returns  $f(\epsilon)$  for holes.
- `ReadhSHETotalDOS`: Returns  $2g_v g(\epsilon)$  for holes.
- `ReadhSHETotalGSV`: Returns  $2g_v g(\epsilon)v^2(\epsilon)$  for holes.

For more information, see [Chapter 39 on page 1207](#).

The parameters for the SHE method are available in the `SHEDistribution` parameter set. [Table 148 on page 862](#) lists the coefficients of models and their default values.

#### Note:

The optical phonon energy  $\epsilon_{op}$  is closely related to the energy grid spacing. Therefore, the same optical phonon energy must be used in the simulation domain.

*Table 148 Default parameters for SHE distribution model*

Symbol	Parameter name	Electrons	Holes	Unit
$\rho$	<code>rho</code>	2.329		$\text{g/cm}^3$
$\epsilon_{sem}$	<code>epsilon</code>	11.7		$\epsilon_0$
$\epsilon_{ins}$	<code>eps_ins</code>	2.15		$\epsilon_0$
$m_c$	<code>m_s</code>	0.26	0.26	$m_0$
$m_n$	<code>m_dos</code>	0.328	0.689	$m_0$
$m_{ins}$	<code>m_ins</code>	0.5	0.77	$m_0$

**Chapter 25: Hot-Carrier Injection**  
**SHE Distribution Hot-Carrier Injection**

*Table 148 Default parameters for SHE distribution model (Continued)*

<b>Symbol</b>	<b>Parameter name</b>	<b>Electrons</b>	<b>Holes</b>	<b>Unit</b>
$\alpha$	alpha	0.5	0.669	eV <sup>-1</sup>
$g_v$	g	6	1	1
$A$	A	1	1	1
$E_{B0}$	E_barrier	3.1	4.73	eV
$\lambda_{ins}$	Lins	$2.0 \times 10^{-7}$	$2.0 \times 10^{-7}$	cm
$\lambda_{sem}$	Lsem	$5.0 \times 10^{-6}$	$1.0 \times 10^{-6}$	cm
$\tau_0$	tau0	$1.0 \times 10^{-12}$	$1.0 \times 10^{-12}$	s
$D_{ac}/c_L$	Dac_c1	$1.027 \times 10^{-5}$	$6.29 \times 10^{-6}$	eVs/cm
$D_{op}$	Dop	$1.25 \times 10^9$	$8.7 \times 10^8$	eV/cm
$\epsilon_{op}$	HbarOmega	0.06	0.0633	eV
$k_0^2$	swv0	0.0	0.0	cm <sup>-2</sup>
	ii_formula1	1	1	
$s_{ii,1}$	ii_rate1	$1.49 \times 10^{11}$	0.0	s <sup>-1</sup>
$s_{ii,2}$	ii_rate2	$1.13 \times 10^{12}$	$1.14 \times 10^{12}$	s <sup>-1</sup>
$s_{ii,3}$	ii_rate3	0.0	0.0	s <sup>-1</sup>
$\epsilon_{ii,1}$	ii_energy1	1.128	1.128	eV
$\epsilon_{ii,2}$	ii_energy2	1.572	1.49	eV
$\epsilon_{ii,3}$	ii_energy3	1.75	1.49	eV

**Chapter 25: Hot-Carrier Injection**  
 SHE Distribution Hot-Carrier Injection

*Table 148 Default parameters for SHE distribution model (Continued)*

Symbol	Parameter name	Electrons	Holes	Unit
v <sub>ii,1</sub>	ii_exponent1	3.0	0.0	1
v <sub>ii,2</sub>	ii_exponent2	2.0	3.4	1
v <sub>ii,3</sub>	ii_exponent3	0.0	0.0	1

**Table 149** lists the coefficients and the default values of the tabulated doping-dependent fitting parameters  $\zeta_{\text{major}}$  and  $\zeta_{\text{minor}}$  for electrons and holes.

**Note:**

By default, the fitting parameters  $\zeta_{\text{major}}$  and  $\zeta_{\text{minor}}$  are neglected as the impurity scattering rate is adjusted automatically according to the low-field mobility. You must switch off `AdjustImpurityScattering` to use these parameters.

*Table 149 Default parameters for unitless doping-dependent functions  $\zeta_{\text{major}}$  and  $\zeta_{\text{minor}}$*

Doping	Parameter name (electrons)	$\zeta_{\text{major}}$ (electrons)	$\zeta_{\text{minor}}$ (electrons)	Parameter name (holes)	$\zeta_{\text{major}}$ (holes)	$\zeta_{\text{minor}}$ (holes)
$10^{15.00}/\text{cm}^3$	e <sub>fit(0)</sub>	1.20698	2.63089	h <sub>fit(0)</sub>	2.36872	3.84998
$10^{15.25}/\text{cm}^3$	e <sub>fit(1)</sub>	1.26585	2.61522	h <sub>fit(1)</sub>	2.47647	3.82989
$10^{15.50}/\text{cm}^3$	e <sub>fit(2)</sub>	1.35031	2.62123	h <sub>fit(2)</sub>	2.65631	3.87730
$10^{15.75}/\text{cm}^3$	e <sub>fit(3)</sub>	1.45972	2.64751	h <sub>fit(3)</sub>	2.91784	3.98847
$10^{16.00}/\text{cm}^3$	e <sub>fit(4)</sub>	1.59727	2.68504	h <sub>fit(4)</sub>	3.28127	4.16424
$10^{16.25}/\text{cm}^3$	e <sub>fit(5)</sub>	1.76810	2.73218	h <sub>fit(5)</sub>	3.77842	4.40187
$10^{16.50}/\text{cm}^3$	e <sub>fit(6)</sub>	1.97625	2.77580	h <sub>fit(6)</sub>	4.44356	4.68485
$10^{16.75}/\text{cm}^3$	e <sub>fit(7)</sub>	2.22278	2.80091	h <sub>fit(7)</sub>	5.29810	4.97515

**Chapter 25: Hot-Carrier Injection**  
 SHE Distribution Hot-Carrier Injection

**Table 149 Default parameters for unitless doping-dependent functions  $\zeta_{\text{major}}$  and  $\zeta_{\text{minor}}$**

Doping	Parameter name (electrons)	$\zeta_{\text{major}}$ (electrons)	$\zeta_{\text{minor}}$ (electrons)	Parameter name (holes)	$\zeta_{\text{major}}$ (holes)	$\zeta_{\text{minor}}$ (holes)
$10^{17.00}/\text{cm}^3$	e fit(8)	2.50474	2.79066	h fit(8)	6.33175	5.21189
$10^{17.25}/\text{cm}^3$	e fit(9)	2.81348	2.72938	h fit(9)	7.48564	5.32107
$10^{17.50}/\text{cm}^3$	e fit(10)	3.13088	2.60729	h fit(10)	8.64257	5.23752
$10^{17.75}/\text{cm}^3$	e fit(11)	3.42620	2.42644	h fit(11)	9.62681	4.93200
$10^{18.00}/\text{cm}^3$	e fit(12)	3.66329	2.20490	h fit(12)	10.2280	4.42987
$10^{18.25}/\text{cm}^3$	e fit(13)	3.82090	1.97450	h fit(13)	10.2758	3.80695
$10^{18.50}/\text{cm}^3$	e fit(14)	3.91451	1.77291	h fit(14)	9.74236	3.16136
$10^{18.75}/\text{cm}^3$	e fit(15)	4.00744	1.63637	h fit(15)	8.78324	2.57856
$10^{19.00}/\text{cm}^3$	e fit(16)	4.21180	1.59940	h fit(16)	7.66672	2.11166
$10^{19.25}/\text{cm}^3$	e fit(17)	4.69302	1.70363	h fit(17)	6.65698	1.78292
$10^{19.50}/\text{cm}^3$	e fit(18)	5.69842	2.01596	h fit(18)	5.94642	1.59808
$10^{19.75}/\text{cm}^3$	e fit(19)	7.63117	2.65859	h fit(19)	5.66599	1.56334
$10^{20.00}/\text{cm}^3$	e fit(20)	11.1923	3.85825	h fit(20)	5.94556	1.70207

## Visualizing Spherical Harmonics Expansion Method

For plotting purposes, the SHE method provides several macroscopic variables that can be obtained from the energy distribution function:

$$n_{\text{SHE}} = 2g_v \int_0^{\infty} g(\varepsilon) f(\varepsilon) d\varepsilon \quad (879)$$

$$T_{n, \text{SHE}} = \frac{2g_v}{n_{\text{SHE}}} \int_0^{\infty} \frac{2\varepsilon}{k} g(\varepsilon) f(\varepsilon) d\varepsilon \quad (880)$$

$$G_{n, \text{SHE}}^{\text{ii}} = 2g_v \int_0^{\infty} \frac{1}{\tau_{\text{ii}}(\varepsilon)} g(\varepsilon) f(\varepsilon) d\varepsilon \quad (881)$$

$$\vec{J}_{n, \text{SHE}} = \frac{2qg_v}{3} \int_0^{\infty} \vec{g}(\varepsilon) v^2(\varepsilon) \nabla f(\varepsilon) d\varepsilon \quad (882)$$

$$\vec{v}_{n, \text{SHE}} = -\frac{\vec{J}_{n, \text{SHE}}}{qn_{\text{SHE}}} \quad (883)$$

where  $n_{\text{SHE}}$ ,  $T_{n, \text{SHE}}$ ,  $G_{n, \text{SHE}}^{\text{ii}}$ ,  $\vec{J}_{n, \text{SHE}}$ , and  $\vec{v}_{n, \text{SHE}}$  are the electron density, the average energy, the avalanche generation rate, the current density, and the average velocity, respectively.

The corresponding keywords for plotting these macroscopic variables are:

```
Plot{
    eSHEDensity           # electron density [/cm^3]
    eSHEEnergy             # average electron energy [K]
    eSHEAvalancheGeneration # electron avalanche generation rate [/cm^3s]
    eSHECurrentDensity/Vector # electron current density [A/cm^2]
    eSHEVelocity/Vector     # electron average velocity [cm/s]
}
```

The corresponding keywords for holes are `hSHEDensity`, `hSHEEnergy`, `hSHEAvalancheGeneration`, `hSHECurrentDensity`, and `hSHEVelocity`.

To plot the position-dependent electron-energy distribution function  $f$  for each kinetic energy grid, specify `eSHEDistribution/SpecialVector` in the `Plot` section:

```
Plot{
    ...
    eSHEDistribution/SpecialVector
}
```

## Chapter 25: Hot-Carrier Injection

Carrier Injection With Explicitly Evaluated Boundary Conditions for Continuity Equations

Similarly, for the hole-energy distribution function, specify `hSHEDistribution/SpecialVector`. The column  $i$  of the special vector represents the distribution function at  $\epsilon = (i + 1)\Delta\epsilon$ . For example, `eSHEDistribution_C2` represents  $f$  at  $\epsilon = 3\Delta\epsilon$ .

In addition, Sentaurus Device allows you to plot the electron-energy distribution function versus kinetic energy at positions specified in the command file.

The plot file is a `.plt` file, and its name must be defined in the `File` section by the keyword `eSHEDistribution`:

```
File {
    ...
    eSHEDistribution = "edist"
}
```

Plotting is activated by including the `eSHEDistributionPlot` section (similar to the `CurrentPlot` section) in the command file with a set of coordinates of positions:

```
eSHEDistributionPlot {
    (-0.02 0) (0 0) (0.02 0)
    ...
}
```

For each position defined by its coordinates, Sentaurus Device determines the enclosing element and interpolates the distribution function using the data at the element vertices. Similarly, for the hole-energy distribution function, define `hSHEDistribution` in the `File` section and include the `hSHEDistributionPlot` section in the command file.

---

## Carrier Injection With Explicitly Evaluated Boundary Conditions for Continuity Equations

### Note:

This feature is available only for transient simulations. It is especially useful for writing and erasing memory cells.

Hot-carrier current can be added as an interface boundary condition for continuity equations in adjacent semiconductor regions. A typical structure (see [Figure 53](#)) consists of sequences of semiconductor–insulator–semiconductor regions.

Hot-carrier current produced at one semiconductor–insulator interface from the sequence is added to the second semiconductor–insulator interface, using the closest vertex algorithm described in [Destination of Injected Current on page 845](#).

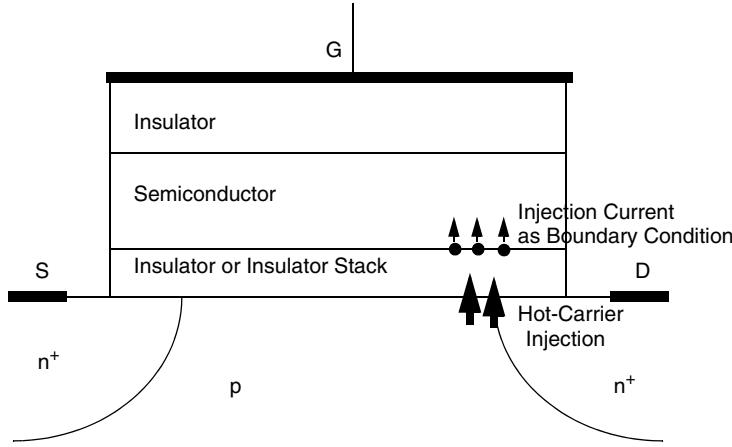
At each time step after the solution is computed, the hot-carrier injection (HCl) currents are post-evaluated using the solution. For the next time step, the HCl currents are added using the current boundary condition for continuity equations in semiconductor regions, where carriers are injected, and then the whole carrier transport task is solved self-consistently.

## Chapter 25: Hot-Carrier Injection

### Carrier Injection With Explicitly Evaluated Boundary Conditions for Continuity Equations

The carriers leave the semiconductor region where they are produced and enter nonlocally into the semiconductor region where they are injected. To conserve current, injection current is subtracted from the former semiconductor region and added to the latter.

Figure 53 *Injection of hot-carrier current in a MOSFET structure*



By using this method, the solution is obtained self-consistently (with one time-step delay) even if there is no carrier transport through the insulator region.

The feature is activated automatically during a transient simulation when any of the hot-carrier injection models is activated in the `GateCurrent` section and the floating semiconductor region where the hot carriers are to be injected does not have a charge boundary condition specified. Specifying `GateName` in the `GateCurrent` section disables the feature.

In addition, hot-carrier injection and the currents of the Fowler–Nordheim tunneling model can be computed at semiconductor–oxide-as-semiconductor interfaces. This is an extension of the searching algorithm described in [Destination of Injected Current on page 845](#). In this case, there is a semiconductor–semiconductor interface instead of semiconductor–insulator interface. To avoid ambiguity, one of the interface regions must be selected as an insulator using the keyword `InjectionRegion`:

```
Physics(RegionInterface="Region_sem1/Region_sem2") {
    GateCurrent(Fowler eLucky InjectionRegion="Region_sem2")
}
```

---

## References

- [1] K. Hasnat *et al.*, "A Pseudo-Lucky Electron Model for Simulation of Electron Gate Current in Submicron NMOSFET's," *IEEE Transactions on Electron Devices*, vol. 43, no. 8, pp. 1264–1273, 1996.
- [2] C. Fiega *et al.*, "Simple and Efficient Modeling of EPROM Writing," *IEEE Transactions on Electron Devices*, vol. 38, no. 3, pp. 603–610, 1991.
- [3] S. Jin *et al.*, "Gate Current Calculations Using Spherical Harmonic Expansion of Boltzmann Equation," in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, San Diego, CA, USA, pp. 202–205, September 2009.
- [4] A. Gnudi *et al.*, "Two-dimensional MOSFET Simulation by Means of a Multidimensional Spherical Harmonics Expansion of the Boltzmann Transport Equation," *Solid-State Electronics*, vol. 36, no. 4, pp. 575–581, 1993.
- [5] C. Jacoboni and L. Reggiani, "The Monte Carlo method for the solution of charge transport in semiconductors with applications to covalent materials," *Reviews of Modern Physics*, vol. 55, no. 3, pp. 645–705, 1983.
- [6] C. Jungemann and B. Meinerzhagen, *Hierarchical Device Simulation: The Monte-Carlo Perspective*, Vienna: Springer, 2003.
- [7] E. Cartier *et al.*, "Impact ionization in silicon," *Applied Physics Letters*, vol. 62, no. 25, pp. 3339–3341, 1993.
- [8] T. Kunikiyo *et al.*, "A model of impact ionization due to the primary hole in silicon for a full band Monte Carlo simulation," *Journal of Applied Physics*, vol. 79, no. 10, pp. 7718–7725, 1996.
- [9] S.-M. Hong and C. Jungemann, "A fully coupled scheme for a Boltzmann-Poisson equation solver based on a spherical harmonics expansion," *Journal of Computational Electronics*, vol. 8, no. 3-4, pp. 225–241, 2009.